Table of Contents

1. Copyright ........................................................................................................................................... 1
2. Introduction of Windows clients in opsi ................................................................................................. 2
  2.1. Conventions of this document ........................................................................................................ 2
3. Requirements for Windows clients ......................................................................................................... 3
  3.1. Windows versions supported by opsi ............................................................................................... 3
  3.2. Hints to select the right client hardware .......................................................................................... 3
    3.2.1. General information .................................................................................................................. 3
    3.2.2. Ethernet / RJ45 / USB-Adapter / MacPassThrough ................................................................. 4
    3.2.3. Wake on Lan / Wake on Bios ...................................................................................................... 5
    3.2.4. BIOS maintenance ....................................................................................................................... 5
    3.2.5. MAC-addresses ............................................................................................................................ 6
    3.2.6. System memory (RAM) ................................................................................................................ 6
    3.2.7. Hardware test ............................................................................................................................... 6
    3.2.8. Driver provisioning ..................................................................................................................... 6
    3.2.9. Requirements table .................................................................................................................... 6
4. Installing the minimal Windows opsi products ...................................................................................... 8
  4.1. Automatic installation of the minimal Windows opsi products .......................................................... 8
  4.2. Manual installation of the Windows opsi products .......................................................................... 9
5. Adding clients to opsi ............................................................................................................................ 10
  5.1. Creation of a new opsi client ............................................................................................................... 10
    5.1.1. Creating a new opsi client via the graphical management interface ..................................... 10
    5.1.2. Creating a new opsi client via the command line .................................................................... 10
    5.1.3. Creating a new opsi client using the opsi-client-bootcd .......................................................... 11
  5.2. Integration of existing clients ........................................................................................................... 13
    5.2.1. Using the installer ...................................................................................................................... 13
    5.2.2. Using opsi-deploy-client-agent ............................................................................................... 14
6. Rollout of existing products .................................................................................................................. 16
  6.1. Deploying opsi standard products: opsi-configed ............................................................................ 16
  6.2. Inventory with the localboot products hwaudit and swaudit ............................................................ 16
7. Management interface opsi-configed ..................................................................................................... 18
  7.1. Installation of the management interface opsi-configed ................................................................. 18
  7.2. Start of the management interface opsi-configed ........................................................................... 18
8. Hardware Inventory with the netboot product hwinvent ...................................................................... 19
9. Installation of a new Windows PC with opsi (OS Installation) .............................................................. 20
  9.1. OS-Installation: Complete the Base Package for Windows .............................................................. 20
  9.2. NT6 family: Win7 / 2008R2 and up .................................................................................................. 20
    9.2.1. Creating a PE ............................................................................................................................ 21
    9.2.2. Extending a PE .......................................................................................................................... 22
    9.2.3. unattend.xml ............................................................................................................................ 23
    9.2.4. Driver Integration ...................................................................................................................... 24
    9.2.5. Providing the Installation Files ................................................................................................. 24
    9.2.6. Installation Log files .................................................................................................................. 24
  9.3. Windows Product Key ..................................................................................................................... 24
  9.4. Start the Windows Installation ......................................................................................................... 25
  9.5. Structure of the Unattended Installation Products ......................................................................... 25
    9.5.1. Directory Tree Overview ......................................................................................................... 25
    9.5.2. File Descriptions ....................................................................................................................... 26
    9.5.3. Directory installfiles / winpe .................................................................................................... 26
1. Copyright

The Copyright of this manual is held by uib gmbh in Mainz, Germany.

This manual is published under the creative commons license 'Attribution - ShareAlike' (by-sa).

A description of the license can be found here:
https://creativecommons.org/licenses/by-sa/3.0/

The legally binding text of the license can be found here:
https://creativecommons.org/licenses/by-sa/3.0/legalcode

Most parts of the opsi software is open source.
Not open source are the parts of the source code which contain new extensions, that are still under cofunding, which have not been paid off yet. See also: opsi cofunding projects

All of the open source code is published under the AGPLv3.

The legally binding text of the AGPLv3 license can be found here: http://www.gnu.org/licenses/agpl-3.0-standalone.html

Information about the AGPL: http://www.gnu.org/licenses/agpl-3.0.en.html

For licenses to use opsi in the context of closed source software, please contact uib gmbh.

The names 'opsi', 'opsi.org', 'open pc server integration' and the opsi logo are registered trademarks of uib gmbh.
2. Introduction of Windows clients in opsi

This manual describes the operation of Windows clients in opsi.

It is assumed that the installation and commissioning of an opsi-server has already been completed.

Main topics of this manual:

1. Automatic Windows OS installation
2. Enrollment and integration of Windows computers in opsi (Installation of the opsi-client-agent)
3. Provision of the standard opsi software for Windows on the opsi server
4. Installation of standard software on the Windows clients
5. Standard opsi software for Windows under opsi
6. Packaging your own software
7. Creation of opsi packages
8. Notes on Windows clients
   a. Special commands for Windows
   b. Directories that you can use

2.1. Conventions of this document

Commands are highlighted separately:

This is a command

As part of the installation and configuration, you can usually copy the commands from these fields one after the other using copy & paste from this document and execute them.

This is opsi-script code:

Message "Installing "$ProductId$" ...

Chapters that contain the name of a particular platform are specific to that platform. The supported platforms are:

- Windows
- Linux
- macOS
3. Requirements for Windows clients

In the following the requirements for the management of Windows clients under opsi are described.

Technical requirements are an opsi-server with opsi 4.1.

### 3.1. Windows versions supported by opsi

<table>
<thead>
<tr>
<th>Windows Version</th>
<th>Opsi 4.2</th>
<th>Opsi 4.1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows 11</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>Windows 2022</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>Windows 10</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>Windows 2019</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>Windows 2016</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>Windows 2012 R2</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>Windows 8.1</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>Windows 2012</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>Windows 8</td>
<td>🚷</td>
<td>🚷</td>
</tr>
<tr>
<td>Windows 2008 R2</td>
<td>✔️</td>
<td>🚷</td>
</tr>
<tr>
<td>Windows 7</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>Windows 2008</td>
<td>🚷</td>
<td>🚷</td>
</tr>
<tr>
<td>Windows Vista</td>
<td>🚷</td>
<td>🚷</td>
</tr>
<tr>
<td>Windows 2003</td>
<td>🚷</td>
<td>🚷</td>
</tr>
<tr>
<td>Windows XP</td>
<td>🚷</td>
<td>🚷</td>
</tr>
<tr>
<td>Windows 2000</td>
<td>🚷</td>
<td>🚷</td>
</tr>
</tbody>
</table>

- ✔️: Supported
- 🚷: Unsupported
- 🚷: Under development
- ▲: Discontinued

### 3.2. Hints to select the right client hardware

#### 3.2.1. General information

There are no lists of supported or unsupported hardware for opsi, and due to the fast pace of the market, it will not be possible to maintain such lists.

Nevertheless, some requirements should be observed when procuring client hardware. This is not because of specific problems with opsi, but because of the fundamental challenges of client management. See also: Scheduling installations

In general, enterprise class devices are more likely to be suitable than consumer devices.

General information on the procurement of client hardware can be found here:

- [https://www.itk-beschaffung.de/EN/Guidelines/Notebooks](https://www.itk-beschaffung.de/EN/Guidelines/Notebooks)
- [https://www.itk-beschaffung.de/EN/Guidelines/PC-Desktop](https://www.itk-beschaffung.de/EN/Guidelines/PC-Desktop)
3.2.2. Ethernet / RJ45 / USB-Adapter / MacPassThrough

A wired network connection is of great importance or a necessity for many functions of client management. If possible, a client with a network adapter / RJ45 interface on the device is preferable to an adapter-based solution.

* Figure 1. Ethernet RJ45 connector

(Source / License of image: https://commons.wikimedia.org/wiki/File:Ethernet_RJ45_connector_p1160054.jpg
https://creativecommons.org/licenses/by-sa/3.0/deed.en)

- Booting from network
  No network boot without a wired network.
  Without network boot, operating system installations and some other features are not available.
  See also:
  - Netboot products
  - ops-local-image
  - ops-clonezilla

- Wake on Lan
  The possibility of starting a computer via the network usually requires a wired network.

- Bandwidth
  WLAN access points now reach remarkable bandwidths. In the case of installations on many devices on the same access point, these devices have to share the bandwidth of the access point, which in the best case leads to long installation times and, in the worst case, errors and failures.
  Therefore, wired network connections are still preferable for client management when possible.

Therefore, newly procured devices should have a wired network connection when possible.
However, more and more devices are marketed without a cable connection.
Should (for whatever reason) a device be obtained without Ethernet on board, a cable connection via an (USB) adapter or (USB) docking station should be possible and the following further prerequisites should be observed.

* Figure 2. USB-RJ45-Adapter

Such USB-RJ45 adapters bring the following problem: These ‘adapters’ are actually external network cards connected via USB. This means that the MAC address of this network interface does not identify the device but the adapter. This means that a fixed assignment of a MAC address to a device is in effect no longer possible. However, this fixed assignment is required for network booting and Wake-on-LAN.

This problem is known to the manufacturers and ‘MacPassthrough’ is offered as a solution for this.
3. Requirements for Windows clients

3.2.3. Wake on Lan / Wake on Bios

The device should support Wake on Lan.
Ideally, no special drivers should be necessary for this. Otherwise it can happen that a newer MS driver is loaded as part of a Microsoft update and Wake-on-LAN fails until the special driver is installed again.
For devices without ethernet onboard, the Wake-on-LAN should ideally also work via the USB-RJ45 adapter with the address provided by 'MacPassThrough'.

If there is no wired network available or if this is not connected in normal operation, it should be possible to wake the computer on days and times that can be entered in the BIOS (Wake on Bios / Wake Up Timer).

3.2.4. BIOS maintenance

From the preceding it is made clear that the configuration of the BIOS is an important step in client management. Therefore, the configuration and maintenance of the BIOS should be script-controlled and non-interactive. This means that you can carry out the necessary configurations of the BIOS via an opsi-product:

- BIOS update
- Set / change BIOS password
- Switch UEFI / Secureboot on / off
- Switch on Wake-on-LAN
- Switch on 'MacPassThrough'
- Switch wake timer on / off and set times

The script-controlled configuration should also be possible with the BIOS password switched on.
Attention: Even if a script-controlled BIOS configuration is possible in principle, it does not necessarily have to include all BIOS options. Therefore, make sure that the important options are all configurable.

Technically speaking, the possibilities for the BIOS configuration are usually based on an extension of the WMI of the computer, which can then be addressed via WMI or PowerShell.
For more details here a few links:

- https://docs.lenovocdrt.com/#/tbct/tbct_top
3.2.5. MAC-addresses

If you buy a larger number of devices, integration of the devices into your environment is easier if you know the MAC-addresses. Therefore, the supplier should be able to provide you with a machine-readable list of MAC-addresses.

3.2.6. System memory (RAM)

For a network boot, the opsi-linux-bootimage must be loaded and started in the system memory (RAM). At least 2 GB RAM is currently required.
For some Linux distributions (e.g. Ubuntu / Mint), 4 GB is required.

3.2.7. Hardware test

Before you buy a larger number of devices, it is always a good idea to request a reference device, which you can then test with opsi. You can also commission such a test from uib gmbh.

3.2.8. Driver provisioning

Ideally, the manufacturer should provide model-specific driver packages on its website. These driver packages must then be 'extract / unpacked' so that the drivers for this model can be easily made available on the opsi server.
It is an advantage if no separate PE-driver is required for the wired network interface, so the same driver can be used in WinPE and for the installed Windows OS.

3.2.9. Requirements table

Wired network (RJ45):

• on board (better) or via USB-adapter with 'MacPassThrough' support in the BIOS
• Network boot via IPv4 / IPv6
• Wake-on-LAN support
  - Also with default drivers?
  - Also for use with MacPassThrough?

Script-controlled BIOS configuration:

• BIOS Update
• Set / change BIOS password
• Switch UEFI / Secureboot on / off
• Switch on Wake-on-LAN
• Switch on 'MacPassThrough'
• Switch wake timer on / off and set times

Miscellaneous:

• Driver provisioning / extractable?
3. Requirements for Windows clients

- Driver provisioning / PE driver needed?
- MAC Address list
- Enough RAM for the bootimage (2 GB / 4 GB for Linux)
4. Installing the minimal Windows opsi products

To distribute software with opsi, ready-made products are available for installation. These include the agent (‘opsi-client-agent’), which must be installed on the clients for management.

There is an automated and a manual way to do this. The automated option is recommended.

4.1. Automatic installation of the minimal Windows opsi products

For the automatic installation of the opsi products there is the tool `opsi-package-updater`, which configured as in `/etc/opsi/opsi-package-updater.conf` or `/etc/opsi/package-updater.repos.d/`, automatically fetches the current packages from the opsi repository and installs them on the server.

The configuration of the opsi repository for Windows clients can be found in the directory `/etc/opsi/package-updater.repos.d/` in the file `uib-windows.repo`.

Activate the desired repository by setting the entry `active = true` in the desired *.repo file.

Listing 1. `/etc/opsi/package-updater.repos.d/uib-windows.repo`

```plaintext
[repository_uib_windows]
description = opsi Windows Support
active = true
baseUrl = http://download.uib.de
autoInstall = false
autoUpdate = true
autoSetup = false
; Set Proxy handler like: http://10.10.10.1:8080
proxy =
```

Install the packages on the server by running this command as root:

```
opsi-package-updater -v --repo uib_windows install
```

After a successful installation you have to reload all data in `opsi-configed` so that the new products are visible there.

If the connection to the Internet has to be routed via a proxy, this must be entered in the `.repo` configuration files in `/etc/opsi/package-updater.repos.d/` as the value for `proxy`. From version 4.1.1.33 of opsi-utils a global proxy can be configured in `/etc/opsi/opsi-package-updater.conf`.

```
[repository_uib_windows]
...
proxy =
```

If the installed packages are to be updated later, this can be linuxed with the following command:
Further information on `opsi-package-updater` can be found in the manual.

Please note that OS installation products such as win10-x64 are not immediately ready for use after installation. The installation has to be supplemented by the installation files of the corresponding installation medium (see: OS-Installation: Complete the Base Package for Windows).

### 4.2. Manual installation of the Windows opsi products

There is also the option of downloading and installing the packages manually.


We recommend saving the `.opsi` files under `/var/lib/opsi/repository`. To ensure that the process `opsiconfd` can access the files, you should run `opsi-set-rights /var/lib/opsi/repository`.

After downloading you have to install the packages on the server with the command `opsi-package-manager -i <packagename>.opsi`.
5. Adding clients to opsi

To be able to manage computers with opsi, they must be known to the opsi system. In addition, an agent must be running on these computers so that communication between the server and client is possible. No management is possible without this client agent.

Depending on the environment in which opsi is to be used, there are different procedures. If there are already clients in the environment with an installed operating system that are to be managed with opsi, they can be integrated in different ways.

The alternative to this is that the computers to be managed by opsi are equipped with a new operating system. As part of the installation of the operating system, the required agent is also installed by opsi. However, any previously installed software (including the operating system) will be removed. To use this procedure you first add a client to opsi and then perform an OS installation.

5.1. Creation of a new opsi client

To manage computers, they must be known to the opsi-server. This chapter describes different ways to create a client in opsi for later management. This is particularly helpful if you want to install an operating system on your computer using opsi.

For the integration of clients with an already installed operating system, please read the chapter integration of existing Clients.

5.1.1. Creating a new opsi client via the graphical management interface

A client can be added to the opsi-server through the opsi-configed graphical user interface.

From the menu, choose OpsiClient / Create new opsi client and enter:

- Client name
- DNS domain (if different from the default)
- Client description
- IP address (required if DNS can not be used resolve the address of the client)
- MAC address (required if the opsi-server is the DHCP server or if you want to use PXE boot with this client)

After completing the input, the client will be created on the opsi-server, and if the opsi-server is also the DHCP server, the client will also be created in the DHCP configuration, as a PXE client.

The list of configured opsi clients can be viewed at any time in the opsi-configed mode Client configuration under the clients tab.

5.1.2. Creating a new opsi client via the command line

A client can be added through the command line using the tool opsi-admin.

The syntax is the following:

```bash
opsi-admin -d method host_createOpsiClient <client-id> [opsiHostKey] [description] [notes] [hardwareAddress] [ipAddress] [inventoryNumber] [oneTimePassword] [created] [lastSeen]
```

Missing values usually use a default value - most fields are then empty.

The following command will create the client testclient.domain.local with a random host key, the description Testclient, no notes, the MAC address of 00:0c:29:12:34:56 and the IP address 192.0.2.1:
5.1.3. Creating a new opsi client using the opsi-client-bootcd

On the download page of uib you will find various ISO images of the 'opsi-client-boot-cd' at https://download.uib.de/opsi4.2/boot-cd/. Download the latest and burn it to a CD.

Start the computer from the CD. You then should see the following screen:

![Start image opsi-client-boot-cd](image)

*Figure 4. Start image opsi-client-boot-cd*

Choose *Start opsi (English)*. After a while, the following screen will appear. If your DHCP server assigns IP addresses to unknown DHCP clients, then most fields will already have valid values. Otherwise you have to complete the missing data by hand. You must at least give the hostname.
Then choose OK.

Then choose Admin account. This tells the client to register itself at the opsi-server using provided credentials.

Now you will get a login window, where you must authenticate yourself as a member of the opsiadmin group. If this was successful, then the client sends its data to the server, at which point the client will be created at the server. In the next step, the client asks the server for the list of available netboot products, and makes them available for you to choose from.
Now you may choose the operating system that you would like to install (or e.g. hwinvent).

5.2. Integration of existing clients

To include existing clients in opsi, the opsi-client-agent (or opsi-linux-client-agent/opsi-mac-client-agent) must be installed on them. This can be realised in several ways. After you have installed the opsi-client-agent as described below, the client will also appear in the client list of opsi-configed, unless you have already added the client there.

Basically there is the possibility to install the agent on the client or to start the installation from the server.

Executing the installation directly on the client is suitable for individual computers. For a mass rollout of the agent, have a look at opsi-deploy-client-agent. If there is already another way to distribute software available, then it is also possible to distribute the opsi-client-agent by using the opsi-client-agent-installer or the opsi-client-agent MSI-package.

Once the agent is installed, available opsi products can be installed on these clients.

5.2.1. Using the installer

1. Logon to the client.

2. Download the installer from your configserver. It is located at https://<fqdn_or_ip_of_the_configserver>:4447/public/opsi-client-agent/ and has the file name opsi-client-agent-installer.exe (or opsi-linux-client-agent-installer.sh/opsi-mac-client-agent-installer.sh)

3. Execute the installer (for linux and macos this must be done with root-rights, on windows a UAC-Request may be displayed)

4. The installer will extract itself into a temporary local directory and start the oca-installation-helper, which shows a user interface with input fields for Client-ID, Opsi Service URL, Username and Password. The fields are pre-filled, but you may need to add or change some of the data.

5. Client-Id should be the fqdn of the Client. Opsi Service url should have the format https://<fqdn_or_ip_of_the_configserver>:4447. Username and Password should correspond to a user of the group opsiadmin in case of a first installation. For reinstallation it is also possible to use Client-Id and pckey for authentication.

6. The installer connects to the server to register the client at the server. Afterwards the installer calls the included opsi-script to execute the setup.opsiscript of the opsi-[linux-|mac-]client-agent.

When calling the installer, you can supply parameters which enable a fully automatic execution of the installation. Details can be
5.2.2. Using opsi-deploy-client-agent

The `opsi-deploy-client-agent` program installs the opsi-client-agent (or opsi-linux-client-agent/opsi-mac-client-agent) directly from the opsi-server on the clients. This makes it easy to integrate a large number of clients from a server into an opsi environment.

Requirements for Windows clients:

- an open C$ share
- an open admin$ share
- an administrative account
- `winexe` must not be blocked by an antivirus program.
- The program `winexe` must be available on the server. It is part of the `opsi-windows-support` package.

Requirements for Linux clients:

- ssh-access with a user that can perform "sudo" or is root itself.

The `opsi-deploy-client-agent` program can be found at `/var/lib/opsi/depot/opsi-client-agent` (or opsi-linux-client-agent, opsi-mac-client-agent).

Execute the script with 'root' privileges.

The program creates the client on the server, then copies the installation files and the configuration information, including the pckey, to the client. After copying the necessary information, `opsi-deploy-client-agent` starts the installation on the client by calling oca-installation-helper (non-interactive).

With the `opsi-deploy-client-agent` you can also install to a list of clients. To do this, either any number of clients can be passed as the last parameter or the clients can be read from a file using the `-f` option. When using a file, there must be a client on every line.

The program can work with IP addresses, hostnames or FQDNs. It will try to automatically detect what type of address it is processing.

Possible parameters can be found by using `--help:`

Deploy opsi client agent to the specified clients. The c$ and admin$ must be accessible on every client. Simple File Sharing (Folder Options) should be disabled on the Windows machine.

positional arguments:
  host                  The hosts to deploy the opsi-client-agent to.

optional arguments:
  -h, --help            show this help message and exit
  --version, -V         show program’s version number and exit
  --verbose, -v         increase verbosity (can be used multiple times)
  --debug-file DEBUG_FILE  Write debug output to given file.
  --username USERNAME, -u USERNAME  username for authentication (default: Administrator). Example for a domain account: -u <DOMAIN><\<username>
  --password PASSWORD, -p PASSWORD  password for authentication
  --use-fqdn, -c        Use FQDN to connect to client.
  --use-hostname        Use hostname to connect to client.
  --use-ip-address      Use IP address to connect to client.
  --ignore-failed-ping, -x  try installation even if ping fails
  --reboot, -r          reboot computer after installation
  --shutdown, -s        shutdown computer after installation
  --start-opsiclientd, -o  Start opsiclientd service after installation without performing Events (default).
  --no-start-opsiclientd  Do not start opsiclientd service after installation (deprecated).
  --hosts-from-file HOST_FILE, -f HOST_FILE  File containing addresses of hosts (one per line). If there is a space followed by text after the address this will be used as client description for new clients.
  --skip-existing-clients, -S  skip known opsi clients
  --threads MAX_THREADS, -t MAX_THREADS  number of concurrent deployment threads
  --install-timeout INSTALL_TIMEOUT  timeout for single threads (default is unlimited)
  --depot DEPOT  Assign new clients to the given depot.
  --group GROUP  Assign fresh clients to an already existing group.
  --smbclient  Mount the client’s C$-share via smbclient.
  --mount  Mount the client’s C$-share via normal mount on the server for copying the files. This imitates the behaviour of the ‘old’ script.
  --keep-client-on-failure  If the client was created in opsi through this script it will not be removed in case of failure. (DEFAULT)
  --remove-client-on-failure  If the client was created in opsi through this script it will be removed in case of failure.
  --failed-clients-file FAILED_CLIENTS_FILE  filename to store list of failed clients in
6. Rollout of existing products

For the rollout of software on clients, the ‘opsi-client-agent’ must be installed on them. This can be rolled out to existing computers. With an operating system installation via opsi, the ‘opsi-client-agent’ is installed automatically.

In the following, the opsi-configed management interface is used to distribute software to clients.

The following products are provided by opsi for Windows as standard:

- opsi-linux-client-agent
- swaudit
- hwaudit
- l-system-update
- opsi-configed
- opsi-logviewer
- opsi-auto-update
- opsi-linux-client-kiosk
- opsi-setup-detector

6.1. Deploying opsi standard products: opsi-configed

One of the opsi standard products is the product opsi-configed, which installs the opsi Management Interface. This Application is a Java application, therefore a Java Runtime Engine is bundled with the product.

Using opsi-configed, in the mode Configuration of clients, choose the appropriate client in the tab Clients.

If you have not already done so, update the data of opsi-configed by using File / Reload all data or click the reload icon.

Switch to the tab Product configuration, look for the line with the product opsi-configed. Click in the column Requested Action, and select the action setup.

The check mark in the icon menu bar should change its color to red. If you click on it, the new settings will be transmitted to the opsi-server, afterwards its color will be green again.

Reboot the client. The opsi-client-agent should start and install the product opsi-configed. After the installation you can find opsi-configed in the start menu.

6.2. Inventory with the localboot products hwaudit and swaudit.

In opsi-configed, Client configuration mode, under the Clients tab, select the client under consideration.

If not already performed, update the opsi-configed’s dataset using Reload File/Data or clicking the corresponding icon.

Go to the Product configuration tab, click in the Requested column for the hwaudit product, this will open a list/dropdown menu and there select the setup action. Repeat this for the swaudit product.

The check mark in the icon menu bar should change its color to red. If you click it, the new settings will be transmitted to the opsi-server, afterwards its color will be green again.

Then restart the client. It should now start the opsi-client-agent and install the hwaudit and swaudit products. With hwaudit and swaudit, hardware and software information, respectively, is collected and transmitted to the opsi-server. The collected information
is displayed under the *Hardware Information* and *Software Inventory* tabs, respectively.
7. Management interface opsi-configed

Opsi offers with the opsi-configed a comfortable management interface. It communicates via HTTPS with the opsi server and can therefore be used on any computer that can establish a corresponding connection.

When using a virtual machine, make sure that the virtual screen has a large enough resolution. For opsi-configed a minimum resolution of 1024x768 pixels is required. To improve the graphics and mouse driver integration at a higher resolution, it is helpful to install the 'VMware Tools' on a VMware machine or the virtual guest additions on a VirtualBox machine.

7.1. Installation of the management interface opsi-configed

The management interface is installed as a local application on the administration PCs. In your web browser, go to the address https://<opsidepotserver>:4447/. There you will find links to installers for different operating systems.

Alternatively, you can find corresponding installers under https://download.uib.de/opsi4.2/misc/helper/.

The Windows installer must be executed with administrative rights. To do this, right click to open the context menu of the installer and then select 'Run as administrator'.

Once one PC is equipped with the management interface, further PCs can have easily have the interface Section 6.1, “Deploying opsi standard products: opsi-configed” installed with the localboot product opsi-configed, as long as the opsi agent is already installed on the PC.

7.2. Start of the management interface opsi-configed

Start opsi-configed via the shortcut in your Start menu.

Log in as a user who is a member of the group opsiadmin.

The operation of the management interface is pretty much self explanatory. You will find detailed instructions in the opsi manual.

Changes in the opsi management interface must be saved before they take effect and changes in the data must be retrieved from the server via the 'Reload data' button.
8. Hardware Inventory with the netboot product hwinvent

If the product 'hwinvent' is installed on your opsi server and you have added a client Section 5.1, “Creation of a new opsi client” which is configured to boot over the network, you can do something else useful: Hardware inventory when there is no operating system installed.

Using 'opsi-configed', in the mode 'Configuration of clients', choose the appropriate client in the tab 'Clients'. If you have not already done so, update the data of opsi-configed by using 'File / Reload all data' or click the reload icon. Switch to the tab 'Netboot products', look for the line with the product hwinvent. Click in the column 'Requested Action', and select the action 'setup'. The check mark in the icon menu bar should change its color to red. If you click on it, the new settings will be transmitted to the opsi-server, afterwards its color will be green again.

Then reboot the client. It should now pull a Linux image over the network (via PXE), to scan the hardware of the PC and then reboot it. If the computer was not otherwise already set up, afterwards the message appears that no operating system is installed on the disk.

The results of the hardware scan have been transmitted to the opsi-server. The results can be viewed under the 'Hardware information' tab.

In case the screen remains black after booting the bootimage or if the network card does not work (correctly), the start parameters of the bootimage may have to be adjusted for this specific hardware.

This can be achieved using 'opsi-configed' in the 'Host parameters' tab by editing the entry 'opsi-linux-bootimage.append'.

More information can be found in the opsi manual, in the chapter 'netboot products'.
9. Installation of a new Windows PC with opsi (OS Installation)

The following describes how a computer with no operating system can get a Windows OS installed with opsi.

Suitable clients are real or virtual computers with at least 2048 MB RAM and a network card with network boot support: This means that they support the PXE protocol for booting systems via the network. The network boot has to be activated in the BIOS menu or moved to the first position of the bootorder options.

Virtual hardware is usually well supported by the Windows standard drivers, which can be tried if perform a test installation of Windows. To install Windows on newer real-world machines, you may need to integrate additional drivers first. For an initial test, you can use a VMware Appliance that contains an empty machine and can run in VMware Workstation Player.

For the following chapter you should create a corresponding client in opsi Section 5.1, “Creation of a new opsi client”. This can be done easily through opsi-configed.

Some tools useful for deploying Windows with opsi are installed through the 'opsi-windows-support' package.

9.1. OS-Installation: Complete the Base Package for Windows

The opsi win-OS-packages contain only the files that are necessary to perform our automated OS installation, but not the operating system software itself.

For an automatic installation of a Windows operating system, you have to copy your existing original Windows installation files (and if necessary, store the Windows license key on the server).

9.2. NT6 family: Win7 / 2008R2 and up

In order to perform an OS Installation, a so-called WinPE is being used as a 'Live OS'. You can create it using an opsi package (opsi-winpe), or manually if you so desire. Generally speaking, the Windows version of the PE is independent of the Windows OS version being installed. Above all, the availability of drivers for disk- and network devices is important. Microsoft recommends a 32-Bit PE for 32-bit installations, and a 64-Bit PE for 64-bit installations.

"To install a 64-bit version of Windows you must use a 64-bit version of Windows PE. Likewise, to install a 32-bit version of Windows, you must use a 32-bit version of Windows PE."

In any case you will need an "Assessment and Deployment Kit" (ADK, Windows 8.1 or 10), or its predecessor "Windows Automated Installation Kit" (Windows AIK; until Windows 7), which you install on a supported (preferably 64-bit) Windows OS:

- Windows 10 / 8.1 ADK

Install the Windows PE Add-On for ADK (if possible on a 64-bit machine) in the suggested path under Program Files (x86). Select only the "Windows Pre-Installation Environment (Windows PE)"; Dependencies are automatically selected.

- link: WAIK Windows 7

This site provides you with an ISO file, which may then be burned to a CD or mounted, and then installed.
9.2.1. Creating a PE

The simplest method requires a computer that has opsi-client-agent installed, as well as the Windows ADK (Win8.1, Win10). The manual method is described below in Section 9.2.1.2, “Manual PE creation for Windows 10 & Windows 8 (ADK).”

Automated PE creation using opsi

- Using opsi-configed set the localboot-product `opsi-winpe` to `once` for the client you intend to use, if desired adjust the product property to `x86` instead of `x64` at the lower right side, and save (right click > save).
- If the opsi-product `opsi-winpe` is missing, install it onto your opsi-server with the command `opsi-package-updater -v install opsi-winpe`.
- Launch an installation event for the client (right click > on-demand, or reboot).
- After a successful completion of this action, move or copy the contents of the now existing folder on your client `C:\winpe_<ARCH>\media\` into the pre-existing folder within the OS folder you want to install: `\\opsiserver\opsi_depot_rw\<operating system>\winpe\`
- Finally run the following command on the console of your opsi server. Finished.

```
opsi-set-rights
```

Manual PE creation for Windows 10 & Windows 8 (ADK)

The console commands are very similar in 32- or 64-bit versions, except for the `<ARCH>` entries. These have to be replaced with either `x86`, `amd64` or `ia64`.

Run Start ⇒ “Windows Kits” ⇒ “Windows ADK” ⇒ “Deployment and Imaging Toolkits Environment” from the Start Menu. A command prompt will open which has the required environment variables set.

- Copy the WinPE

```
copype.cmd <ARCH> C:\winpe
```

- Mount the Image

```
dism /Mount-Wim /WimFile:C:\winpe\media\sources\boot.wim /index:1 /MountDir:c:\winpe\mount
```

- replace startnet.cmd

```
echo c:\opsi\startnet.cmd > "c:\winpe\mount\Windows\System32\startnet.cmd"
```

(Note: The file `c:\opsi\startnet.cmd` will be created by the opsi linux bootimage after the script `setup.py` is executed. The `startnet.cmd` contains the call to wpeinit.)

- Unmount the Image

```
dism /Unmount-Wim /MountDir:c:\winpe\mount /Commit
```

- Copy the contents of `C:\winpe\ISO` to `/var/lib/opsi/depot/<productid>/winpe`.

Adjust the access rights by entering:
Manual PE creation for Windows 7 (WAIK)

The console commands are very similar in 32- or 64-bit versions, except for the `<ARCH>` entries. These have to be replaced with either `x86`, `amd64` or `ia64`.

Start a command prompt as Administrator with elevated rights (Start ⇒ Programs ⇒ Accessories ⇒ right click on "Command Prompt" ⇒ "Run as" ⇒ Administrator).

- Copy the WinPE

  "%ProgramFiles%\Windows AIK\Tools\PETools\copype.cmd" <ARCH> C:\winpe

- Mount Image:

  "%ProgramFiles%\Windows AIK\Tools\<ARCH>\imagex.exe" /mountrw "C:\winpe\winpe.wim" 1 "C:\winpe\mount"

- replace startnet.cmd

  echo c:\opsi\startnet.cmd > "C:\winpe\mount\Windows\System32\startnet.cmd"

  (Note: The file c:\opsi\startnet.cmd will be created by the opsi linux bootimage after the script setup.py is executed. The startnet.cmd contains the call to wpeinit.)

- Unmount the Image

  "%ProgramFiles%\Windows AIK\Tools\<ARCH>\imagex.exe" /commit /unmount "C:\winpe\mount"

- Move the WinPE now (From this target dir more files will be moved to the server).

  move "C:\winpe\winpe.wim" "C:\winpe\ISO\sources\boot.wim"

- Copy the contents of `C:\winpe\media` to `/var/lib/opsi/depot/<productid>/winpe`. Adjust the access rights by entering:

  opsi-set-rights /var/lib/opsi/depot/<productid>/winpe

9.2.2. Extending a PE

In some cases it is useful to extend a PE. Especially when using Dell-Hardware. Dell provides special network and storage drivers specially recommended for use in a PE. These instructions only work with Windows 7. (Windows Vista does not inherit the needed DISM- Deployment Image Servicing and Management.) These instructions assume that you have already completed the previous chapter and have created a PE.

The Windows Automated Installation Kit is not needed for following instructions.
9. Installation of a new Windows PC with opsi (OS Installation)

The first step is to download Dell-PE-drivers from the Dell-Website. For Windows 7, you will need the WINPE 3.0 Drivers from Dell. The downloaded CAB-File must be extracted to the local disk. This can be done with 7-zip or the command-line-tool Expand.exe. For simplicity, we recommend creating a directory called “dell-driver” on the local disk, and then extracting the CAB-File into this directory.

• First dism is used to scan the image, in order to determine the required index number. Start a command prompt as administrator (Start ⇒ Programs ⇒ Accessories ⇒ right click on “Command Prompt” ⇒ “Run as” ⇒ (Administrator) and run the following command:

```bash
dism /Get-WimInfo /WimFile:C:\winpe\ISO\sources\boot.wim
```

In the output of this command, you can see which images are included in the image file. Normally a PE-image is a one-image-file, so you can generally use the index 1, but it is better to check first.

• The next command mounts the image for modification:

```bash
dism /Mount-Wim /WimFile:C:\winpe\ISO\sources\boot.wim /index:1 /MountDir:c:\winpe\mount
```

• To integrate the extracted drivers into the mounted image, you need to execute this command:

```bash
dism /Image:C:\winpe\mount /Add-Driver /Driver:c:\dell-driver\winpe\x64 /Recurse
```

If the architecture is 32-bit, the `x64` must be replaced with `x86`. The Driver-CAB from Dell contains the drivers for both architectures.

If only one driver has to be integrated, then leave out the option `/Recurse`, and point directly to the driver-inf-file instead of the driver-directory. Furthermore, with the option `/ForceUnsigned` it is possible to integrate unsigned drivers to the image.

• Finally the image is unmounted, and the changes are committed:

```bash
dism /Unmount-Wim /MountDir:c:\winpe\mount /Commit
```

• Copy the contents of `C:\winpe\ISO` to `/var/lib/opsi/depot/<productid>/winpe`. Adjust the access rights by entering:

```bash
opsi-set-rights /var/lib/opsi/depot/<productid>/winpe
```

9.2.3. unattend.xml

The control file for the unattended installation is the XML file `unattend.xml`, which you can find under `/var/lib/opsi/depot/win7/custom`. Any modifications to this file should be made in this directory and not in the opsi directory.

The file `unattend.xml` that comes with the opsi package, contains references to the netboot productproperties, which among other things is responsible for activating the Administrator account with the password ‘nt123’.

Documentation for `unattend.xml` can be found in the directory `C:\Program Files\Windows\Waik\docs\chms`, after installing the WAIK.
9.2.4. Driver Integration

The driver integration proceeds as described here: Section 9.6, “Simplified Driver Integration during the unattended Windows Installation”.

9.2.5. Providing the Installation Files

Copy the complete installation DVD to /var/lib/opsi/depot/<productid>/installfiles And adjust the rights and ownership:

```
opsi-set-rights /var/lib/opsi/depot/<productid>/installfiles
```

9.2.6. Installation Log files

- c:\Windows\Panther\setupact.log:
  Log until the end of setup phase 4 (running under WinPE)
- c:\Windows\Panther\setupact.err:
  Error log until the end of setup phase 4 (running under WinPE)
- c:\Windows\Panther\UnattendGC\setupact.log:
  Log from the specialize phase
- c:\Windows\Panther\UnattendGC\setupact.err:
  Error log from the specialize phase
- c:\Windows\System32\Winevt\Logs\*
- c:\Windows\ntbtlog.txt (only when startup logging is activated)

9.3. Windows Product Key

If you have the opsi license management module, you can manage the Windows product keys using the license management module. Read the license management manual or the corresponding chapter in the opsi manual.

If you do not have the license management module, or do not want to use it, proceed as follows.

If you have already set up opsi clients, you can enter a Windows product key per client in the opsi configuration editor:

- select a client
- switch to the netboot products tab
- select the product (e.g. win10-x64)
- change the product property productkey in the lower right corner
- enter the key in the value field
- save by clicking on the “red tick” and leave the field
- save the changes in the backend (“red tick” at the top right).

Or you can assign a default for the Windows product key for the complete opsi depot, which can also be done via the opsi configuration editor:

- Select the depot properties in the configuration editor (tile top right)
- Switch to the Product Default Properties tab
- select the product (e.g. win10-x64)
9.4. Start the Windows Installation

To start a Windows installation, select the relevant client in opsi-configed, set in the 'Netboot products' tab the action to 'setup' for the desired operating system (e.g. win10-x64). Click on the red checkmark (which turns green again).

The client should now load the opsi-linux-bootimage via the network when booting, where you have to confirm the new OS installation again. Then everything should continue automatically until the logon prompt of the installed Windows is finally on the screen.

If the screen remains black after loading the boot image or the network card does not work correctly, the start parameters of the boot image may have to be adjusted for this specific hardware. You can do this in 'opsi-configed' in the 'Host parameters' tab at the entry 'opsi-linux-bootimage.append'. You can find details on this in the opsi manual in the 'Netboot Products' chapter.

Beware of clients with a hard disk larger than 2 TB. In a non-UEFI system, the maximum partition size is 2 terabytes. If a larger partition is to be created, the installation will fail. This a technical limitation of the standard partition table. You need to split the hard drive into partitions. You can control this via the product properties. Or you can purchase the UEFI module, which eliminates this technical limitation.

9.5. Structure of the Unattended Installation Products

This chapter applies to the Windows netboot products.

9.5.1. Directory Tree Overview
9.5.2. File Descriptions

- **setup.py**
  This is the installation script which is executed by the boot image.

- **<productid>_<version>.control**
  Contains the metadata of the product as prepared from the package maintainer. These files are here for information purposes only. Changes to this file have no effect on the system.

- **<productid>.files**
  This file is created automatically and should not be changed.

- **create_driver_links.py**
  show_drivers.py
  These scripts are for driver integration, which is explained in more detail in the chapter Simplified driver integration in the automatic Windows installation.

9.5.3. Directory installfiles / winpe

- **installfiles**
  This directory contains all files from the installation CD/DVD.

- **winpe**
  Contains a bootable winpe image.

9.5.4. Directories opsi and custom

These two directories contain scripts and configuration files for controlling the operating system installation. During installation, priority is given to files in the custom directories.

The opsi directory contains files that can be overwritten without notice by updates. So no changes to these files should be made. For adjustments, you can make changes in the directory custom, which is preserved during updates.
The subdirectory `postinst.d` contains scripts which are started via the `postinst.cmd` after the actual installation of the operating system, e.g. to install the opsi-client-agent. The scripts are processed in alphabetical order. To clarify the order of execution, the file names begin with a two-digit number (`10_dhcp.cmd`). If you want to make extensions here, you can store scripts in the `custom/postinst.d` directory with starting numbers between decades (`13_myscript.cmd`). The starting numbers 10, 20, 30,... are reserved for maintenance by opsi.org/uib. The script `99_cleanup.cmd` is the final script and ends with a reboot.

### 9.5.5. Directory drivers

This directory is used for the integration of drivers and is described in the following chapter.

### 9.6. Simplified Driver Integration during the unattended Windows Installation

When managing a group of PCs that have devices whose drivers are not included in the standard Windows installation, it usually makes sense to integrate these drivers directly into the installation. In the case of network devices, this can sometimes be unavoidable, because a Windows without a network card is not easily accessible for the administrator.

Opsi supports the automatic integration of drivers into the installation, and therefore simplifies driver deployment. The drivers simply need to be placed into the correct directory. By executing a script, the driver directories are searched and a catalog is created, based on which the bootimage can automatically identify and integrate the correct drivers. Standard drivers, USB drivers, HD audio drivers as well as drivers for hard disk controllers (text mode drivers) can be stored and automatically integrated.

In order for the drivers to be installed with the Windows installation, they must be stored in a specific form on the server. Suitable drivers contain a `*.inf` file that describes the driver for the Windows Setup program. Any drivers in `setup.exe`, `*.zip` or packed any other way are not usable. If you have a computer that already has the drivers installed, then you can extract the drivers in the correct format with the program ‘double driver’ ([http://www.boozet.org/dd.htm](http://www.boozet.org/dd.htm)).

There are multiple levels of driver integration:

- General driver packages
- Drivers that are suitable for your hardware but are not specially assigned
- Drivers that are manually assigned to computers
- Drivers that are automatically assigned to the computers via the `<vendor>/<model>` fields of the inventory.

How these different levels can be used is described below:

#### 9.6.1. General Driver Packages

When the hardware configuration across the computers is very heterogeneous, then it can make sense to work with general driver packages. General drivers can be placed under `./drivers/drivers`.

Drivers which are found in `./drivers/drivers/`, will be matched to the corresponding hardware using the PCI IDs (or USB- or HD_Audio-IDs) in the description file, and then integrated into the Windows setup if needed.

#### 9.6.2. Drivers that suitable for your hardware but not specially assigned

In case you have to support few different hardware configurations, you can use the drivers provided by the manufacturers. Additional or tested drivers belong in their own directories (name and depth of the directory structure do not matter) below the directory `./drivers/drivers/preferred`.

Drivers located in the directory `./drivers/drivers/preferred` are prioritised over the drivers in `./drivers/drivers/` by using the PCI IDs (or USB- or HD_Audio-IDs) in the description file, and then integrated into the Windows setup if needed.

Problems can occur when the same PCI ID can be found in the description file of different drivers in `preferred`. In this case a direct
assignment of the drivers to the client is necessary.

### 9.6.3. Drivers manually assigned to clients

Additional drivers that are to be installed regardless of their assignment or detection via the PCI- or USB-IDs must be in their own directories (name and depth of the directory structure are irrelevant) below the directory `.drivers/drivers/additional`. Via the product property `additional_drivers` you can assign one or more paths of driver directories within `.drivers/drivers/additional` to a client. Directories specified in the `additional_drivers` product property are searched recursively and all included drivers will be integrated. Symbolic links are also followed. You can use this to create a directory for certain computer types (e.g. dell-optiplex-815).

If a driver for a matching PCI device (or HD audio, USB) is found in the driver directories specified via `additional_drivers`, then no other driver from `drivers/preferred` or `drivers/` is integrated for this device (`additional_drivers` can be thought of as `superverified`). This means that `additional_drivers` has the function of adding drivers that would not be found via normal driver detection.

### 9.6.4. Drivers automatically assigned to the clients using the inventory fields

The mechanism of direct assignment of drivers to devices described in the previous section can be automated since opsi 4.0.2. The directory `.drivers/drivers/additional/byAudit` is searched for a directory name that corresponds to the `vendor` found during hardware inventory. A search is now made in this `vendor` directory for a directory name that corresponds to the `model` found during hardware inventory. If such a directory is found, this directory is treated as if it were manually assigned via the product property `additional_drivers`. The directory name `byAudit` is case sensitive. The directory names for `Vendor` and `Model` are not case sensitive (`Dell` and `dELL` are treated the same way).

Since opsi 4.0.5, the drivers for a opsi-client can be made available via opsi-configed in the Hardware Inventory tab (see: opsi manual "Automatic driver upload").

The opsi-linux-bootimage looks for drivers in the order:

- `<vendor>/<model> (<sku>)`
- if in the previous no match is found `<system vendor>/<system model>` is checked.
- if in the previous no match is found `<motherboard vendor>/<motherboard model>` is checked.

Some manufacturers use model names, which are very unfavourable for this method, because you can not use some special characters such as `/` in file- or directory names. An example of this would be a model name like: “5000/6000/7000”. A directory with this name is not permitted due to the special characters. Since opsi 4.0.3 the following special characters: `< > ? " : | `/ *` have therefore been replaced internally by a `_`. With this change you can create the directory for the example as: “5000_6000_7000” and the directory is automatically assigned, although the information in the hardware inventory does not correspond to the directory structure.

### 9.6.5. Structure of the Driver Directory and Driver Files
9.6.6. Processing of the Different Levels of Driver Integration

The top priority is to include all drivers that are found using the property ‘additional_drivers’ or using the inventory data in ./drivers/drivers/additional/byAudit. As part of the integration of drivers, it is checked for which hardware of a device (based on the PCI-, USB-, HD-Audio IDs) a driver has been made available in this way. Only for devices that are not matched by a driver, the following methods are used in order to find a matching driver.

For devices for which a driver has not been assigned via ‘additional_drivers’ (or ‘byAudit’), a suitable driver is searched for and integrated using the PCI ID (or USB-, HD-Audio ID).

‘Integration’ of drivers means the following:

- The driver will be copied to the local hard drive at c:\drv\<num>.
- The Windows Setup is told in the unattended file to search for matching drivers in c:\drv\.

9.6.7. Add and check drivers

After adding a driver or any other change in the ./drivers/drivers directory (or below), execute the following command in the root directory of the netboot product directory to set the rights correctly:

```
opsi-set-rights ./drivers
```

After storing drivers in the directories ./drivers/drivers or ./drivers/drivers/preferred, then run the script ./create_driver_links.py. The script searches the directories under './drivers/drivers' and generates a list of links that can be used to identify the assignment of the drivers to specific hardware (PCI-IDs, USB-IDs and HD-Audio-IDs). The script will prioritize the drivers in the preferred directories.

The script setup.py of the bootimage examines the hardware of the computer to be installed and identifies the necessary drivers. These are then copied to the hard disk and the unattend.xml will be patched accordingly.

If a hardware inventory is available for a client, you can use the command:

```
./show_drivers.py <clientname>
```
This will show which drivers the boot image would choose for installation via PCI-IDs, USB-IDs, HD-Audio-IDs and 'additional_drivers' (or 'byAudit') and for which hardware no driver is available yet.

Use the output of `show_drivers.py` to check if the desired drivers will be integrated.

It is possible that driver directories from manufacturers contain drivers for different operating system versions (e.g. Windows 7/8.1/10) or different configurations (SATA / SATA-Raid). This cannot be differentiated automatically. If you suspect that the wrong driver will be used, move this driver to the `drivers/exclude` directory and then run `create_driver_links.py` again. Drivers in the directory 'drivers/exclude' are not used during driver integration.

Example output of `show_drivers.py` for a client:

```bash
./show_drivers.py pcdummy

PCI-Devices
[(Standardsystemgeräte), Standard PCI to PCI bridge]
   No driver - device directory /var/lib/opsi/depot/<productid>/drivers/pciids/1022/9602 not found
[ATI Technologies Inc., Rage Fury Pro (Microsoft Corporation)]
   Using build-in windows driver
[(Standard-IDE-ATA/ATAPI-Controller), Standard-Dual-Channel-PCI-IDE-Controller]
/var/lib/opsi/depot/<productid>/drivers/drivers/D/M/N/123
[Realtek Semiconductor Corp., Realtek RTL8168C(P)/8111C(P) PCI-E Gigabit Ethernet NIC]
/var/lib/opsi/depot/<productid>/drivers/drivers/preferred/realtek_gigabit_net_8111_8168b
[IEEE 1394 OHCI-conform Hostcontroller-Manufacturer, OHCI-conform IEEE 1394-Hostcontroller]
   No driver - device directory /var/lib/opsi/depot/<productid>/drivers/pciids/197B/2380' not found
[Advanced Micro Devices, Inc., AMD AHCI Compatible RAID Controller]
   /var/lib/opsi/depot/<productid>/drivers/drivers/preferred/ati_raid_sb7xx
[(Standard-USB-Hostcontroller), Standard OpenHCD USB-Hostcontroller]
   No driver - device directory /var/lib/opsi/depot/<productid>/drivers/pciids/1002/4397' not found
[ATI Technologies Inc, ATI SMBus]
   /var/lib/opsi/depot/<productid>/drivers/drivers/preferred/ati_smbus

USB-Devices
[(Standard-USB-Hostcontroller), USB-Connection device]
/var/lib/opsi/depot/<productid>/drivers/drivers/preferred/brother_844x_pGerb
[Microsoft, USB-Printersupport]
   /var/lib/opsi/depot/<productid>/drivers/drivers/preferred/brother_844x_pGerb

Additional drivers
[ati_hdaudio_azalia]
/var/lib/opsi/depot/<productid>/drivers/drivers/additional/ati_hdaudio_azalia
```

Example for a client with 'additional_drivers':

```bash

```

Example for a client with 'additional_drivers':

```bash

```
PC-Devices

**[8086:27C8]** Intel : Intel(R) N10/ICH7 Family USB Universal Host Controller - 27C8
/var/lib/opsi/depot/<productid>/drivers/drivers/preferred/R293337/WIN7

**[8086:27DA]** Intel : Intel(R) N10/ICH7 Family SMBus Controller - 27DA
/var/lib/opsi/depot/<productid>/drivers/drivers/preferred/R293337/WIN7

**[8086:27C9]** Intel : Intel(R) N10/ICH7 Family USB Universal Host Controller - 27C9
/var/lib/opsi/depot/<productid>/drivers/drivers/preferred/R293337/WIN7

**[8086:27D0]** Intel : Intel(R) N10/ICH7 Family PCI Express Root Port - 27D0
/var/lib/opsi/depot/<productid>/drivers/drivers/preferred/R293337/WIN7

Using build-in windows driver

This driver will not be integrated, because same device already integrated in:
/var/lib/opsi/depot/<productid>/drivers/drivers/not_preferred/x64/C/Intel/1/dmi_pci.inf'

**[8086:27B8]** Intel : Intel(R) ICH7 Family LPC Interface Controller - 27B8
/var/lib/opsi/depot/<productid>/drivers/drivers/preferred/R293337/WIN7

**[8086:27D2]** Intel : Intel(R) N10/ICH7 Family Serial ATA Storage Controller - 27D2
/var/lib/opsi/depot/<productid>/drivers/drivers/preferred/R293337/WIN7

**[8086:27D8]** Microsoft : High Definition Audio-Controller

No driver - device directory '/var/lib/opsi/depot/<productid>/drivers/pciids/8086/27D8' not found

**[10EC:0662]** Realtek : Realtek High Definition Audio

Manually selected [hp_e5800] /var/lib/opsi/depot/<productid>/drivers/drivers/additional/hp_e5800/64bit

HD-Audio-Devices

**[10EC:8662]** Realtek High Definition Audio

Manually selected [hp_e5800] /var/lib/opsi/depot/<productid>/drivers/drivers/additional/hp_e5800/64bit
Example for a client with 'byAudit':

```bash
./show_drivers.py pctry5detlef
Manually selected drivers (additional)
  - /var/lib/opsi/depot/<productid>/drivers/drivers/additional/byAudit/nvidia/awrdacpi
  - /var/lib/opsi/depot/<productid>/drivers/drivers/additional/byAudit/nvidia/awrdacpi/pctry5detlef/Display/Radeon X300-X550-X1050 Series Secondary (Microsoft Corporation - WDDM)/atiilhag.inf
  - /var/lib/opsi/depot/<productid>/drivers/drivers/additional/byAudit/nvidia/awrdacpi/pctry5detlef/Display/Radeon X300-X550-X1050 Series (Microsoft Corporation - WDDM)/atiilhag.inf

PCI-Devices
  [10DE:0058]  NVIDIA : CK804 Ethernet Controller
  Using build-in windows driver
  /var/lib/opsi/depot/<productid>/drivers/drivers/preferred/ga-ma78-pcbon4/chipset_win7-64/SMBUS
  No driver - device directory '/var/lib/opsi/depot/<productid>/drivers/drivers/pciids/104C/8025' not found

/D9A0:E0B70]  ATI Technologies Inc. : Radeon X300/X550/X1050 Series (Microsoft Corporation - WDDM)
  Manually selected
  Multiple selected

/D9A0:E0B71]  ATI Technologies Inc. : Radeon X300/X550/X1050 Series Secondary (Microsoft Corporation - WDDM)
  Multiple selected

/D9A0:E0B72]  AMD : K8 [Athlon64/Opteron] HyperTransport Technology Configuration
  Using build-in windows driver

/D9A0:E0B73]  AMD : K8 [Athlon64/Opteron] Address Map
  Using build-in windows driver

/D9A0:E0B74]  AMD : K8 [Athlon64/Opteron] Miscellaneous Control
  Using build-in windows driver

/D9A0:E0B75]  Realtek : Realtek AC'97 Audio
  Manually selected
  Multiple selected
```

9. Installation of a new Windows PC with opsi (OS Installation) 32 / 76
USB-Devices

[1241:1111] (Standardsystemgeräte): USB-Eingabegerät
No driver - vendor directory '/var/lib/opsi/depot/<productid>/drivers/usbids/1241' not found

HD-Audio-Devices
No devices installed

TIPS

- Directory names such as NDIS1 contain Vista drivers; NDIS2 contain Win7 drivers
- NDIS versions:

NDIS 6.0: Windows Vista
NDIS 6.20: Windows 7, Server 2008 R2
NDIS 6.30: Windows 8, Windows Server 2012
NDIS 6.50: Windows 10, version 1507
NDIS 6.70: Windows 10, version 1703
NDIS 6.80: Windows 10, version 1709
NDIS 6.81: Windows 10, version 1803
NDIS 6.82: Windows 10, version 1809 and Windows Server 2019
NDIS 6.83: Windows 10, version 1903

- Some chipset drivers contain description files, which specify hardware without actually providing drivers. An example would be the cougar.inf or ibexahci.inf from Intel. If such a 'pseudo driver' directory is assigned via 'additional_drivers' (or 'byAudit'), this means that the hardware listed here is excluded from further searches for drivers in the 'preferred' directory.

- SATA drivers and SATA-RAID drivers refer to the same PCI ID. However, a SATA RAID driver will not function with a single-disk system.

- Check the output of ./show_drivers.py carefully!
10. Integration of New Software Packages into the opsi Server

The primary objective of software distribution is to accomplish automatic software installation without user interaction. Software installation and user activity should be strictly separated. In most cases, the installation process requires administrative privileges which the user usually doesn't have. So the installation process has to be done independently from the user. This way, the user can neither interfere nor be affected by the software installation process.

In order to do this, you have to write a script for the script driven installer, which is called an 'opsi-script' script. This script in addition to the installfiles and some metadata can be packed as a opsi-product, which in turn can be installed on a opsi-server.

10.1. A Brief Tutorial: How to write a opsi-script Script

10.1.1. Introduction

This tutorial merely helps you getting started with opsi. It can't replace professional training (which you may order through uib), or thoroughly studying the complete opsi manuals (which might be time consuming and partially error prone if you lack background knowledge). uib now offers training in English, too.

Training and Support:
Get Training by uib gmbh in Europe or possibly Northern America:
https://uib.de/en/support-training/support/

Manuals:
The opsi Manuals can be found at: https://uib.de/en/opsi-documentation/documentation/ important for scripting: opsi-script reference card and opsi-script manual

Wiki (Scripts, Tips, Links):
https://forum.opsi.org/wiki

Support Forum (fast and free vendor support):
https://forum.opsi.org

10.1.2. Methods of Non-Interactive Installation for Windows

Regardless of whether or not you are using opsi or another management solution, there are three different ways to install software without user interaction:

1. Unattended or Silent Installation
   The original setup binary from the software manufacturer can be executed with command line arguments which enable a 'silent' or 'unattended' mode. It depends on whether or not the program supports a silent installation mode. A special case of this method is the unattended installation of MSI packages. "silent" Installation of a MSI-Package: A MSI-Package can be installed using the "quiet" Option.

2. Interactive Setup with recorded Answers
   The actions executed by the administrator while running the manufacturer's setup program during a manual installation are automatically recorded using the free tools 'AutoIt' or 'Autohotkey'. This generates an autoIt script which in turn can be used for an unattended installation.

3. Recreate the setup-routine with opsi-script:
   The actions executed by the setup-routine when installing manually are recorded and the opsi-script is used to reproduce them.
opsi supports all of these variants. Usually a combination of these methods in one script provides the best result. For example, performing the basic installation using the original setup if available, and then doing some customizing by patching the registry or the file based configuration.

10.1.3. Structure of an opsi-script script

In the subsequent chapters the basic elements of a opsi-script script will be described with examples for Windows.

First an example for a simple opsi-script script:

```
[actions]
WinBatch_tightvnc_silent_install

[WinBatch_tightvnc_silent_install]
"%ScriptPath\\tightvnc-1.3.9-setup.exe" /silent
```

An opsi-script script consists of primary and secondary sections. Sections are introduced with a section name in square brackets, as known from the ini files. The true software installation work takes place in the secondary sections, which are called by the primary sections.

The secondary sections are “theme specific” and each has a special syntax. The section name of a secondary section starts with its type, followed by a freely determinable name.

In the shown example, the primary section [Actions] calls a secondary section [WinBatch_tightvnc_silent_install]. The secondary section is of the type WinBatch. The content of a WinBatch section is executed via the Windows API. So in this case the setup program tightvnc-1.3.9-setup.exe is started with the parameter /silent.

10.1.4. Primary Sections

**Actions**

The [Actions] section is the actual main program. This is where the script processing begins.

**Sub-sections**

Program sections that are required constantly can be redistributed to sub sections (subroutines). It’s possible to source sub sections to external files.

The primary sections are the main program in which the script flow is controlled. For this there are:

- Variables: Strings and string lists
- if elseif else endif statements
- for loops over string lists
- Functions
10. Integration of New Software Packages into the opsi Server

10.1.5. Important secondary sections

Files

File operations, such as:

- copy (with version control, recursive ...).
- delete
- create directories
- ...

WinBatch

Used to call programs via the Windows API. For example, invokes to setup programs in silent mode are made in these sections.

ShellInAnIcon

The content of this section is passed to the operating system on the typical shell for execution. This shell is the cmd.exe for Windows, for Linux and for macOS the bash. As such, normal batch scripts can be stored here.

Name variants of ShellInAnIcon with identical behavior are Shellbatch, DOSBatch and DOSInAnIcon.

ExecWith

The contents of these sections are passed to an external program such as an (interpreter) for execution. For example, ExecWith can be used to integrate AutoIt scripts http://www.autoitscript.com directly into the opsi-script script.

Registry

The Registry sections are used to edit the registry.

LinkFolder

LinkFolder sections are used to create and remove shortcuts. For example, such shortcuts can be created on the desktop or in the start menu.

10.1.6. Global constants

Global constants are text placeholders that can be used in primary and secondary sections and are textually replaced by their values at runtime.

The use of placeholders can be utilized to ensure that paths are set correctly in different environments (in the case of systems with different languages or operating system versions for example).
Examples:

%ProgramFiles32Dir%
   c:\Program Files (x86)

%Systemroot%
   c:\windows

%System%
   c:\windows\system32

%opsiTmpDir%
   c:\

%Scriptpath%
   <path to running script>

10.1.7. Second example: tightvnc

For clarification purposes, now a simple script for the installation of tightvnc. As a matter of fact this script would get on with the call of the silent installation in the Winbatch section. However, during a repeated installation an interactive dialog appears here (because of the restart of the running service). This dialog window is closed (if it appears) with the help of 'AutoIt'.

```
[Actions]
Message "Installing tightvnc 1.3.9 ..."
ExecWith_autoit_confirm "%ScriptPath%\autoit3.exe" WINST /letThemGo
WinBatch_tightvnc_silent_install
KillTask "autoit3.exe"

[WinBatch_tightvnc_silent_install]
"%ScriptPath%\tightvnc-1.3.9-setup.exe" /silent

[ExecWith_autoit_confirm]
; Wait for the confirm dialog which only appears if tightvnc was installed before as service
; Waiting for the window to appear
WinWait("Confirm")
; Activate (move focus to) window
WinActivate("Confirm")
; Choose answer no
Send("N")
```

10.1.8. Elementary commands for primary sections

**String-Variable**

**Variable declaration**

'DefVar <variable name> [= <initial value>]'

**Variable assignment**

'Set <variable name> = <value>'

**Example:**
DefVar $ProductId$
Set $ProductId$ = "firefox"

or

DefVar $ProductId$ = "firefox"

String variables are handled differently in primary and secondary sections. In primary sections, string variables are independent objects. Only here they can be declared and assigned values. Accordingly, the connection of variables and strings to a string expression is to be performed with a "+" operator.

Example: "Installing "$ProductId$" ...

In secondary sections, string variables are replaced with the contents of the variable before the section is executed.

For example: "Installing $ProductId$ ...

This should be taken into consideration when the corresponding string expressions are cut and pasted in the script.

The advantage of this construction is that in sections that are executed outside the 'opsi-script' (DosBatch / Execwith) opsi-script variables can be manipulated at ease.

**Message / ShowBitmap**

**Text output during installation:**

Message <string>

**Example:**

Message "Installing "$ProductId$" ..."

**To output a graphic during the installation:**

ShowBitmap <filename> <subtitle>

**Example:**

ShowBitmap "%ScriptPath%\python.png" "Python"

**if [elseif] [else] endif**

**Syntax:**

if <condition>
 ;statement(s)
[elseif <condition>
 ;statement(s)]
[
 else
 ;statement(s)
]
endif
Functions

HasMinimumSpace
Checks for free space on the hard disk.

FileExists
Checks for the existence of a file or directory.

Errors, logging and comments

Comment characters ‘;’
Lines that start with a semicolon (‘;’) are not interpreted.

Comment
Writes a comment message to the log file.

LogError
Writes an error message to the log file.

IsFatalError
Cancels the execution of the running script and reports the installation as failed.

Condition for execution

requiredWinstVersion
specifies the (minimum) required opsi-script version.

Other important opsi-script functions

An overview of the opsi-script functions is given by the reference card

A detailed documentation can be found in the opsi-script manual:

Here are a few more notes on particularly important elements:

Stringlists:
String lists are very powerful, especially for evaluating output from external programs. Read the opsi-script docs for more information.

ExitWindows:
Reboot/Shutdown the system and finish the opsi-script.

- ExitWindows /Reboot
  Computer restart after completion of the running script.
- ExitWindows /ImmediateReboot
  Immediate reboot.
- ExitWindows /ImmediateLogout
  Immediately stop script editing and terminate opsi-script.

Product-Properties:
For some products it’s necessary to provide options. These are specifically evaluated per client at runtime. How such properties are created is described in the chapter Creation of opsi product packages
The access to the values of the properties is done via the function `GetProductProperty`:

```plaintext
if GetProductProperty("example-property", "no") = "yes"
   Files_copy_extra_files
endif
```

**Encoding:**

Write your scripts in UTF-8 encoding and set the line `encoding=utf8` at the beginning of the file.

**Special commands for Windows**

- `getOS : string // returns: Linux or Windows_NT or MacOS [W/L/M]
- `getMsVersionInfo : string // Windows Version Information [W]
- `GetMsVersionName //Windows marketing Version //since 4.12.4.35 [W]
- `getMSVersionMap : stringlist [W]

**10.1.9. Example: Windows-Template 'opsi-template'**

This template can be created with `opsi-setup-detector`.

**Listing 2. setup.opsiscript: Installationsscript**

```plaintext
; -------------------------------
; This is a opsi-script file.
; See https://opsi.org https://uib.de
; This code was originally created by opsi-setup-detector 4.2.0.10
; -------------------------------
encoding=utf8

[Actions]
requiredOpsiVersion >> "4.12.4.23"
importlib "uib_exitcode.opsiscript"
importlib "osd-lib.opsiscript"

DefVar $ProductId$
DefVar $InstallDir$
DefVar $MinimumSpace$
DefVar $ExitCode$
DefVar $ErrorString$
DefVar $LicenseRequired$
DefVar $LicenseKey$
DefVar $LicensePool$
DefVar $OS$
DefVar $MsiId$
DefVar $UninstallProgram$
```
;  Please edit the following values:
;  $ProductId$ is the name of the product in opsi, only lower letters, no umlauts, no white spaces, use '-' as a seperator
Set $ProductId$ = "opsi-template"
Set $MinimumSpace$ = "1 MB"
; the path were we find the product after the installation
Set $InstallDir$ = "%ProgramFiles32Dir%\<path to the product>"
Set $InstallDir$ = "unknown"
Set $LicenseRequired$ = "false"
Set $LicensePool$ = ""

set $OS$ = GetOS
if not(($OS$ = "Windows_NT"))
  logError "Installation aborted: wrong OS version: only Windows"
  isFatalError "wrong OS"
endif

if not(HasMinimumSpace (%SystemDrive%, $MinimumSpace$))
  LogError "Not enough space on %SystemDrive%, " + $MinimumSpace$ + " on drive %SystemDrive% needed for " + $ProductId$
  isFatalError "No Space"
  ; Stop process and set installation status to failed
endif

comment "Show product picture"
ShowBitmap "%ScriptPath%" + $ProductId$ + ".png" $ProductId$

if FileExists("%ScriptPath%\delsub.opsiscript")
  comment "Start uninstall sub section"
  Sub "%ScriptPath%\delsub.opsiscript"
endif

Message "Installing " + $ProductId$ + " ..."
if $LicenseRequired$ = "true"
  comment "Licensing required, reserve license and get license key"
  set $LicenseKey$ = get_licensekey_byPool($LicensePool$)
endif

comment "Start setup program"
ChangeDirectory "%SCRIPTPATH%"
;----------------------------------------------
Winbatch_install
;----------------------------------------------
set $ExitCode$ = getlastexitcode
if "true" = isMsiExitcodeFatal_short($exitcode$, "true", $ErrorString$ )
  LogError $ErrorString$
  isFatalerror $ErrorString$
else
  Comment $ErrorString$
endif

comment "Copy files"
Files_install /32Bit

comment "Patch Registry"
Registry_Install /32Bit
comment "Create shortcuts"
LinkFolder_install

[Winbatch_install]
; Choose one of the following examples as basis for your installation
; You can use $LicenseKey$ var to pass a license key to the installer
;
; === Nullsoft Scriptable Install System ****************************
%sScriptPath%\Setup.exe" /S
;
; === MSI package ****************************
You may use the parameter PIDKEY=$Licensekey$:
msiexec /i "%ScriptPath%\some.msi" /f "%opsiLogDir%\$ProductId$.install_log.txt" /qb-! ALLUSERS=1 REBOOT=ReallySuppress
;
; === InstallShield + MSI ****************************
Attention: The path to the log file should not contain any whitespaces:
%sScriptPath%\setup.exe" /s /v" /l* %opsiLogDir%\$ProductId$.install_log.txt /qb-! ALLUSERS=1 REBOOT=ReallySuppress"
;
; === Inno Setup ****************************
http://unattended.sourceforge.net/InnoSetup_Switches_EXITCodes.html
You may create setup answer file by: setup.exe /SAVEINF="filename"
You may use an answer file by the parameter /LOADINF="filename"
%sScriptPath%\setup.exe" /sp- /silent /norestart /nocancel /SUPPRESSMSGBOXES

[Files_install]
; Example of recursively copying some files into the installation directory:
;
copy -s "%ScriptPath%\files\*.*" "$InstallDir$"

[Registry_install]
; Example of setting some values of an registry key:
;
openkey [HKEY_LOCAL_MACHINE\Software\$ProductId$]
; set "name1" = "some string value"
; set "name2" = REG_DWORD:0001
; set "name3" = REG_BINARY:00 af 99 cd

[LinkFolder_install]
; Example of deleting a folder from AllUsers startmenu:
;
set_basefolder common_programs
delete_subfolder $ProductId$
;
Example of creating a shortcut to the installed exe in AllUsers startmenu:
;
set_basefolder common_programs
set_subfolder $ProductId$
;
set_link
name: $ProductId$
target: <path to the program>
parameters:
 working_dir: $InstallDir$
 icon_file:
 icon_index:
 end_link
;
Example of creating a shortcut to the installed exe on AllUsers desktop:
;
set_basefolder common_desktopdirectory
Listing 3. delsub.opsiscript: Deinstallations-SubSkript

```plaintext
; This is a opsi-script file.
; See https://opsi.org https://uib.de
; This code was originally created by opsi-setup-detector 4.2.0.10
; ------------------------------------------
encoding=utf8

Message "Check for existing installation of " + $ProductId$ + ": ...

Set $MsiId$ = '{XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX}'
Set $UninstallProgram$ = $InstallDir$ + "\uninstall.exe"

if FileExists($UninstallProgram$)
    comment "Uninstall program found, starting uninstall"
    Winbatch_uninstall
    Sub_check_exitcode_del
endif

if not (getRegistryValue("HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall" + $MsiId$ ,
    "DisplayName","32bit") = "")
    comment "MSI id " + $MsiId$ + ": found in registry, starting msiexec to uninstall"
    Winbatch_uninstall_msi
    Sub_check_exitcode_del
endif

cmment "Delete files"
if not(($InstallDir$ = '') or ($InstallDir$ = 'unknown'))
    Files_uninstall
endif

cmment "Cleanup registry"
Registry_uninstall

cmment "Delete program shortcuts"
LinkFolder_uninstall

[Winbatch_uninstall]
    ; Choose one of the following examples as basis for program uninstall
    ;
    ; === Nullsoft Scriptable Install System
    ; maybe better called as
    ; Winbatch_uninstall /WaitforProcessending "Au.exe" /Timeoutseconds 10
```
; "$UninstallProgram$" /S
; === Inno Setup===========================================================================================================
; "$UninstallProgram$" /silent /norestart /SUPPRESSMSGBOXES /nocancel

[Winbatch_uninstall_msi]
msiexec /x $MsiId$ /qb-! REBOOT=ReallySuppress

[Files_uninstall]
; Example for recursively deleting the installation directory:
;
del -sf "$InstallDir\"

[Registry_uninstall]
; Example of deleting a registry key:
;
deletekey [HKEY_LOCAL_MACHINE\Software\$ProductId$]

[LinkFolder_uninstall]
; Example of deleting a folder from AllUsers startmenu:
;
set_basefolder common_programs
delete_subfolder $ProductId$
;
; Example of deleting a shortcut from AllUsers desktop:
;
set_basefolder common_desktopdirectory
set_subfolder ""
delete_element $ProductId$

[Sub_check_exitcode_del]
set $ExitCode$ = getlastexitcode
if "true" = isMsiExitcodeFatal_short($exitcode$, "true", $ErrorString$)
    LogError $ErrorString$
isfatalerror $ErrorString$
else
    Comment $ErrorString$
endif

;-------------------------------------------------------------------------------
Listing 4. uninstall.opsiscript: Deinstallations-Skript

; This is a opsi-script file.  
; See https://opsi.org    https://uib.de  
; This code was originally created by opsi-setup-detector 4.2.0.10  
; encoding=utf8

[Actions]
requiredOpsiScriptVersion >= "4.12.4.23"
importlib "uib_exitcode.opsiscript"
importlib "osd-lib.opsiscript"

DefVar $ProductId$
DefVar $InstallDir$
DefVar $MinimumSpace$
DefVar $ExitCode$
DefVar $ErrorString$
DefVar $LicenseRequired$
DefVar $LicenseKey$
DefVar $LicensePool$
DefVar $OS$
DefVar $MsiId$
DefVar $UninstallProgram$

; Please edit the following values:

; $ProductId$ is the name of the product in opsi, only lower letters, no umlauts, no white spaces, use '-' as a seperator
Set $ProductId$ = "opsi-template"

; the path were we find the product after the installation
Set $InstallDir$ = "%ProgramFiles32Dir%\<path to the product>"
Set $InstallDir$ = "unknown"
Set $LicenseRequired$ = "false"
Set $LicensePool$ = ""

set $OS$ = GetOS
if not(($OS$ = "Windows_NT"))
    logError "Installation aborted: wrong OS version: only Windows"
    isFatalError "wrong OS"
endif

comment "Show product picture"
ShowBitmap "%ScriptPath%\" + $ProductId$ + ".png" $ProductId$

Message "Uninstalling " + $ProductId$ + " ..."
if FileExists("%ScriptPath%\delsub.opsiscript")
    comment "Start uninstall sub section"
    Sub "%ScriptPath%\delsub.opsiscript"
endif
if $LicenseRequired$ = "true"
    comment "Licensing required, free license used"
    Set $tmpstr$ = FreeLicense($LicensePool$)
endif

10.2. Creating an opsi product

10.2.1. Installation of opsi-setup-detector, opsi PackageBuilder and opsi-logviewer

Installation of the opsi-package-builder

The opsi-package-builder is currently available for Windows and Linux and MacOS.

The installation files/packages of the opsi-package-builder can be found here:

There you will find in the first lines of the post links to the installation files for Windows and Linux and MacOS.

The opsi-package-builder is not made by 'uib' but by Holger Pandel from the opsi-community (thank you!).

The opsi-package-builder open source license:
https://github.com/pandel/opsiPackageBuilder/blob/master/LICENSE_DE

The opsi-package-builder has its own documentation, that is part of the installation.

You can install the opsi-package-builder also via opsi:

The package opsipackagebuilder_wlm belongs to the opsi standard products and should be installed on your opsi-server. If not, use:

```bash
opsi-package-updater install opsipackagebuilder_wlm
```

to install it on the opsi-server.

Installation of the opsi-setup-detector

The opsi-setup-detector is currently available for Windows and Linux and MacOS.

You can install the opsi-setup-detector via opsi:

The package opsi-setup-detector belongs to the opsi standard products and should be installed on your opsi-server. If not, use:

```bash
opsi-package-updater install opsi-setup-detector
```

to install it on the opsi-server.

A setup program to install the opsi-setup-detector without opsi can be found at:
https://download.uib.de/opsi4.2/misc/helper/

The base functionality of the opsi-setup-detector is the same on all supported platforms. While analyzing a installer file some helper programs will be called, that may not available or runnable.

- Inno-Setups will be analyzed with innounpack.exe at Windows.
- wix-setups will be analyzed with dark.exe at Windows.
- .deb or .rpm files will be analyzed with the Linux command line tools.

The opsi product opsi-setup-detector has a dependency on the opsi product opsi-package-builder_wlm. The opsi-setup-detector uses the opsi-package-builder if available, but can for the most part also work without it. The installation of the opsi-package-builder is recommended.
Installation of the opsi-logviewer

The opsi-logviewer is currently available for Windows, Linux and MacOS.

You can install the opsi-logviewer via opsi:

The package **opsi-logviewer** is part of the opsi standard products and should be installed on your opsi-server. If not, with:

```
opsi-package-updater install opsi-logviewer
```

You can install it on the opsi-server.

A setup program to install the opsi-setup-detector on Windows even without opsi can be found at:
https://download.uib.de/opsi4.2/misc/helper/

The opsi product **opsi-logviewer** has a stated dependency to the opsi product **javavm**.

10.2.2. The opsi-setup-detector program for creating a Windows script

Opsi-setup-detector Start and necessary configurations

The opsi-setup-detector can be started from the programs menu and can be found there under **opsi.org**. Under Windows the opsi-setup-detector is also integrated into the context menu of the explorer in order to call the setup program directly for analysis with the right mouse button.

![opsi-setup-detector Configuration](image)

**Figure 10. opsi-setup-detector Necessary configuration on first startup**

After the first start of the opsi-setup-detector a configuration mask appears. The following information is required here:

- **fullname**: (Used for entries in the changelog.txt)
- **email_address**: (Used for entries in the changelog.txt)
- **workbench_path**: Path to the directory in which the opsi packages are going to be created. This is ideally the path to the place
where the \texttt{opsi\_workbench} of your opsi-server is mounted.

After all needed configurations has be done and saved, you will see the startpage.

![opsi-setup-detector Start](image)

**Figure 11. opsi-setup-detector Start**

On the main window, select the desired task and follow the dialogs or select the 'Next step' button.

The offered tasks are grouped by:

- OS independent
- Windows
- Linux
- MacOS
- multi platform

The offered tasks for Windows:

1. \textit{Analyze file and create opsi package}
   Here, a setup file is analyzed and the entire process is run until an opsi package is created. This process is described in the next chapter.

2. \textit{Analyze 2 files (32 / 64 bit) and create opsi package}
   The procedure is the same as in point 1 above with the following differences:
   Two setup programs for the 32 and 64 bit architectures are queried and analyzed. The product gets an additional property: \texttt{install\_architecture} with the possible values: \texttt{32bitonly}, \texttt{64bitonly}, \texttt{both}, \texttt{systemspecific}.

3. \textit{Only analyze single file}
   runs, similarly to point 1 above, only that is aborted after the analysis of the setup program.

4. \textit{Create an opsi package template}
   This point does not ask for a installer file, but creates a opsi template product for Windows with the information from the product configuration is already taken over.
opsi-setup-detector: Analyze file and create opsi package

The workflow is here described using *Analyze file and create opsi package* as an example.

**Figure 12. opsi-setup-detector Start**

After you selected the task, you will get a file selection dialog. Select now the setup file that has to be analyzed. The analyze will start directly after the selection is done.

**opsi-setup-detector: Analyze**

**Figure 13. opsi-setup-detector analysis**
If the analyze found no result, you will get here a *Sorry unknown Installer*.

In this dialog you may choose to abort the create process. You may also choose to continue based on the pattern of a choosable installer type.

If we have a successful analyze, you will see the result page.

---

**Figure 14. opsi-setup-detector Result of the analysis**

- **Detected Setup Type**: Type of detected Installer
- **MST allowed**: 
- **Link** with information about the installer
- **Setup file**: Path and name of the analyzed setup file
- **MST file**: For MSI installers or installers which contain MSI, an MST file can be specified here which will be integrated into the MSI call.
- **Msid**: For MSI installers or installers that contain MSI in the form of product code
- **Software version**: The version of the software to be installed if determinable.
- **Setup file size MB**: Size of the setup file in MB
• **Required space MB**: This value is an estimate of six times the size of the setup file and can be adjusted if necessary.

• **InstallDir**: As far as detected the directory where the software will be installed.

• **Unattended installation command**: The determined command for a non-interactive installation.

• **Unattended deinstallation command**: The determined command for a non-interactive deinstallation.

• **Deinstallations program**: The determined deinstallations program

The values determined here can now be corrected or supplemented if necessary. The button Next Step leads to the first page of the product configuration. The metadata of the opsi product to be created is entered here.

The values determined here can be incorrect and are probably incomplete! After an initial installation, you should definitely check the values of **InstallDir**, deinstallation program and **software version** and adjust them in your script if necessary.

### opsi-setup-detector: Product configuration 1

![Figure 15. opsi-setup-detector Product configuration 1](image)

- **opsi Product ID**: this is the name of the opsi package to be generated and is generated from the product name below, where spaces and other invalid characters are replaced by a ‘-’. The proposed opsi Product ID can of course be changed.

- **Product Name**: the name of the software to install. This may have to be corrected manually.

- **Product Version**: The version number determined from the name of the setup file must probably be corrected manually. It may only contain numbers and periods, since it's used for the versioning of the opsi package.

- **Description**: In this field the product name is given as default and should be completed with further hints, which are then set as product description of the opsi package.

- **License required**: If this checkbox is set, `$LicenseRequired$=true` will be set when patching the opsiscript.

### opsi-setup-detector: Priority and dependencies
Figure 16. opsi-setup-detector Product configuration 2

For normal application software you don't have to do anything here, due to the fact that the default settings 'fit'. You can press the Next Step button.

Otherwise, here is an explanation of the settings that are possible:

**Priority**
- affects the installation order. **Recommended for application software: 0**
- Possible values are between 100 (at the very beginning) and -100 (at the very end). If product dependencies also exist, these will also additionally influence the installation sequence.

**Dependencies**
- Here you can define the dependencies between products.
- If the configuration contains the connection data for your opsi-server, the connection will here be started. If the configuration does not contain the password (for security reasons) you will be asked for the password:
Productid

Productid (identifier) of the product to which a dependency exists.

If there is a connection to the opsi-server, this will be noticed in green letters and you may select the `productid` in the DropDownListBox. If there is no connection, this will be noticed in red letters and you have to write the `productid` in the input field.

Require Mode

You can either request the **Action** setup or (see below) the **State** (installed).

Action or State

For **State**: State that the product to which a dependency corresponds, should have (installed). If there is another status, the product is set to setup.

For **Action**: Action request, which should be set on the product, whereupon there is a dependency (setup).

This control is disabled while creating a Meta Product to avoid sense less changes.

Requirement Type

Installation order. If the product for which there is a dependency must be installed before the installation of the current product, then this is before. If it must be installed after the current product, this is after. If the order doesn't matter then nothing has to be entered here.

This control is disabled while creating a Meta Product to avoid sense less changes.

Hint:

Unfortunately there is currently no generic mechanism for uninstalling product dependencies. The Product Dependency mechanism is only reliable for action: setup and the (before- or after-) setup actions to be triggered and installed status, in the case of a requiredAction: uninstall this leads unfortunately to errors.

Another hint:

The actual installation order is determined by a combination of product dependencies and product prioritization. Details about this can be found in the opsi manual in the chapter 'Manipulating the installation sequence by priorities and dependencies'.

**opsi-setup-detector: Properties**

Here, editable properties (product variables) can be defined for the product.
### Figure 18. opsi-setup-detector Property Editor

<table>
<thead>
<tr>
<th>Field / Function</th>
<th>Description</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Property Name</td>
<td>Name of the product variable</td>
<td>This identifier is displayed in the product configuration in opsi-configed and can be read within the scripts with the function <code>GetProductProperty</code>.</td>
</tr>
<tr>
<td>Property Type</td>
<td>Variable type</td>
<td>Possible values: <code>Text</code> / <code>bool</code></td>
</tr>
<tr>
<td>Multivalue</td>
<td>Determines whether the product variable can take only exactly one or multiple values</td>
<td>Only available for type <code>Text</code></td>
</tr>
<tr>
<td>Editable</td>
<td>Determines whether the default values can be overwritten with new or additional values or not</td>
<td>Only available for type <code>Text</code></td>
</tr>
<tr>
<td>Description</td>
<td>Variable function description</td>
<td>Displayed as tooltip in opsi-configed</td>
</tr>
<tr>
<td>Possible values</td>
<td>Comma separated list of possible input values</td>
<td>If editable is set to “True”, the list can be added later within opsi-configed. Only available for type <code>Text</code></td>
</tr>
<tr>
<td>Default value</td>
<td>Default value</td>
<td>Selection list; Only available for type <code>text</code>: Free text field. Only available for type <code>Multivalue</code>: Multiple selection</td>
</tr>
</tbody>
</table>

**opsi-setup-detector: Product Icon**
Here you can select an icon to be displayed during the installation or you can accept the default icon (cogwheel) with Next step and switch to the next tab...

To select another icon, use the button Open icon directory to select the directory in which you expect to find icons. As a preselection you get a directory of ‘open source’ icons: 128x128, supplied with the opsi-setup-detector. Select a subdirectory and the icons will be displayed.

Now you can select an icon from the display.

After the product configuration is performed, the product can be created.

**opsi-setup-detector: Create product**
Figure 20. opsi-setup-detector create product

- **Path to opsi-workbench** is a drive letter or UNC path where the share `opsi_workbench` of your opsi-server is mounted.

- To the left of the button *Create opsi package* there are three possible options, which refer to the function of the button:
  - **Create Mode** is a selection area where you can specify what happens when creating the package:
    - **Create opsi product files** creates the directory tree for the new opsi package on the selected `opsi workbench` if it does not already exist. The files required for the package will be created or copied.
    - **Create opsi product files and build package** performs the operations described in the first point. Additionally, the *opsi Package Builder* is called to create the opsi package from the created directory tree. The exact processes are determined by the selection field *Build Mode*:
      - *Only build* starts the *opsi Package Builder* without interactive GUI, creates an opsi package from the directory tree via server command `opsi-makepackage` and terminates the *opsi Package Builder* after work is done.
      - *build and install* starts the *opsi Package Builder* without interactive GUI, creates from the directory tree via server command `opsi-makepackage` an opsi package installs the package via server command `opsi-package-manager` and finishes the *opsi Package Builder* after the work is done.

- **Create opsi product files and start interactive package builder** performs the operations listed in the first item. Additionally the *opsi Package Builder* is called interactively. You have to quit it yourself to return to the *opsi-setup-detector*. For installation, configuration and operation of the community project *opsi Package Builder* check [https://forum.opsi.org/viewforum.php?f=22](https://forum.opsi.org/viewforum.php?f=22)

- **Create opsi package** is the button that initiates the package creation. If a package with this name already exists, you will be asked if the files in the existing directory should be backed up or deleted:

10.2.3. The opsi PackageBuilder program to modify a script.

At the first start after the installation the opsi PackageBuilder starts in offline mode, because important configuration data for the connection with the opsi-server is missing.

![Figure 21. opsi PackageBuilder  First Start: Offline Mode]

If the startup does not work this way and the start menu does not respond (observed under Linux / KDE), try it from the command line by specifying a path and confirm the error message that the path was not found:

```
opsipackagebuilder --path /home
```

Initial configuration of the opsi PackageBuilder

To enter the missing configuration data open the Settings.
In the **General** tab please enter the following settings:

- **configserver**: full name (FQDN) of your opsi-configserver (e.g. opsi.mycompany.org).
- **opsiadmin user**: username of a member of the group opsiadmin (preferably your username)
- **opsiadmin password**: the password of the user specified above. This will not be displayed and is stored encrypted. It’s necessary for the opsi PackageBuilder to communicate with the opsi-server.
- **opsi Server Version**: opsi 4.1 or higher
- **opsi Workbench**: /var/lib/opsi/workbench
- **command execution compatibility**: opsi 4.0.4 or newer / Sudo without password
- **User**: your full name (used in changelogs)
- **Email**: your email address (used in changelogs)

In the **Program** tab please enter the following settings:

- **Use existing network drive**: Check the box.
- **Development folder**: Path to the directory where the opsi packages should be created. This is ideally the path to where the opsi_workbench of your opsi server is mounted.
- **script editor**: The script editor of the opsi PackageBuilder is only available for Windows unfortunately.
  - Under Windows leave it with the default settings.
- Under Linux: External editor: `/usr/local/bin/jedit`
  Command line options: (empty)
- On MacOS: External editor: `/Application/jedit`
  Command line options: (empty)

In the Administration tab, we recommend the following setting, deviating from the default

- **Package**: `opsi-makepackage -v`.

Save the settings and restart the opsi PackageBuilder. The opsi PackageBuilder should now no longer report *Offline mode*.

**Install, modify and pack packages with the opsi PackageBuilder.**

**Figure 24. opsi PackageBuilder Settings: Management**

**Figure 25. opsi PackageBuilder Start**

Use *Open package (F2)* and select the directory in which you have created with the `opsi-setup-detector` a package. (e.g.: `w:\newprod2`) The product window opens with different tabs. The default tab is *Package*.

**Figure 26. opsi PackageBuilder Package Tab**
In this tab you see on the left side the general metadata of the opsi product as you have already been explained in Section 10.2.2.4, “opsi-setup-detector: Product configuration 1”.

On the right side you see the script files and next to it the button:

Figure 27. opsi PackageBuilder Edit button

With the button you can invoke the file in the script editor specified in the configuration and modify the script. On Windows this is the script editor of the opsi PackageBuilder.

Figure 28. opsi PackageBuilder Script editor under Windows

Key features:

- Color syntax highlighting.
- "Folding" of source code (optional: compact, with comments)
- Lexical definition customizable (to do this, the editor must be invoked via start menu entry)
- Autocomplete for syntax elements and variables
- Freely definable and reusable code blocks ("snippets")

The core component of the editor is the module Scintilla, which is also used in other well known editors, such as Notepad++. The lexical elements (syntax highlighting and folding) for the representation of the script language valid for opsi are however completely written in AutoIt, since Scintilla does not supply its own representation module for opsi scripts. Because AutoIt is an interpreter language, it’s slower than other editors and is therefore only conditionally suitable for editing very large scripts, especially when source code convolution is switched on. In the settings, however, it’s possible to specify whether the editor is invoke with these functions or not, provided that the call is made directly via the script tree. If the editor is open via the link in the start menu, syntax highlighting and convolution are generally switched off at startup and can be activated via the editor menu “View”.

(The editor can also be open via the command line. More information about the possible command line parameters can be check with the “-help” option).
10.2.4. Testing and improving an opsi-script script

For testing and improving a script / product there are two different variants:

- Testing the created script as 'standalone' i.e. without installing it on the opsi-server and deploying it from there to the client.
- "Integrated" testing of the complete product with installation on the server and deployment on a client.

In both cases it will be assumed here that you have created a project with the opsi-setup-detector.
'Standalone' tests

Start the application opsi-script-gui: with double click.

- Windows: Double-click the file `opsi-script.exe`.
  (When starting the program on a Windows 7 / 10 client, "run as administrator" must be used with the right mouse button). If the `opsi-client-agent` is already installed on your computer, you will find it in `C:\Program Files (x86)\opsi.org\opsi-client-agent\opsi-script\opsi-script.exe`. If not, copy from the share `\<opsiserver\opsi_depot`, from the `opsi-script\windows\x86` directory the content of this directory.

- Linux: Start file `/usr/bin/opsi-script`.

- MacOS: Start the application `/Applications/opsi-script`.

You'll see the following window:

![Figure 34. opsi-script-gui in interactive mode](image)

- With **Select Script** you can select the script you want to run.

- With **Start** you can start the script. With it, the script will be executed on this computer.

- Now open the log file with the `opsi-logviewer` to see how the opsi-script interprets the script.
  Make sure that you can adjust the displayed log level with the slider in the lower right hand corner.

- Open the script `setup.opsiscript` in an editor and make the desired changes (do not forget to save). There are several ways to do this:
  - Open the project in `opsis PackageBuilder` and open the editor from there.
  - In principle you can use any other editor of your choice.
  We recommend the editor `jEdit` with opsi-script syntax highlighting, that you can find in the basic opsi products.
You can now customize and save the script in the editor (you can leave the editor open). Switch to the opsi-script window and start the script again with the Start button (the script does not have to be selected again). View the log modified based on your changes in the script with the opsi-logviewer. (Don’t forget reload via context menu or toolbar button).

In this way, i.e. by repeating the points:
- Customize the script and save it
- Execute script
- Check log
you can gradually tailor your scripts to do what you need.

Hints for solving detailed problems can be found in the next chapter. The chapter after the following explains how to create an opsi product from the scripts created in this manner, which you can install on the opsi-server.

'Integrated' tests

With the 'integrated tests' the whole project is always executed by opsi on a test client. Proceed as follows:

- Open the script setup.opsiscript in an editor and make desired changes (do not forget to save). There are several ways to do this:
  - Open the project in opsi PackageBuilder and open the editor from there.
  - In principle you can also use any other editor.

We recommend the editor jEdit with opsi-script syntax highlighting, that you can find in the basic opsi products.

- Product Packing
  - Variant 1: Open the project in the opsi PackageBuilder and start the packing via the button Pack.
  - Variant 2: Login via terminal (e.g. Putty) to the opsi-server and change to the project directory on the workbench. Pack the product with the command opsi-makepackage.
Install the product on the opsi-server.

- Variant 1: Start the install in the opsi PackageBuilder with the button install.
- Variant 2: Start the install in the terminal in the project directory with the command `opsi-package-manager -i <myproctid_version.opsi>`. Where `<myproctid_version.opsi>` is the filename that was output in the previous step when packing.

Select and start product via `opsi-configed`

1. Select the test client in the tab Clients
2. In the tab Product configuration select the product. If the product is not visible (which is normal after the first installation) reload the data via the menu File / Reload all data or the button on the very left of the toolbar.
3. For the selected product set the action request setup and save.
4. Start the client or start it via context menu on_demand if the client is running.
5. Wait until the product has run through on the client.
   - In the tab Logfiles / instlog inspect the log file to see how the opsi-script interprets the script.
     Note that you can adjust the log level displayed here with the slider in the lower right hand corner.

In this way, repetition of the points mentioned:
- Adaptation of the script and saving
- Pack product
- Install product on the server
- Run product on the client
- check log
  you can gradually customize your scripts to do what you need.

10.2.5. Packing with opsi-makepackage

Afterwards you can pack the product. To do this, go to the root directory of the product and execute `opsi-makepackage`. Now the product will be packed.

It's recommended to create the packages immediately with an associated md5 checksum file. This file is used by `opsi-package-updater` among others to ensure package integrity after package transfer. Such a file is created automatically, but for special usage scenarios its creation can be avoided.

When transferring packages on the opsi-depotserver, ‘zsync’ can be used to transfer only differences between different packages. In order to use this method, a special `.zsync` file is needed. Such a file is created automatically, but for special usage scenarios the creation can be avoided.

If there are space problems in the temporary directory `/tmp` when creating large packages, it's possible to specify an alternate temporary directory using `--temp-directory`.

If a package of this version already exists, `opsi-makepackage` will show a query:

```
Press <O> to overwrite, <C> to abort or <N> to specify a new version:
```

With o you can choose to overwrite, with c you cancel the process and with n you can choose to be asked for a new product or package version.

You can install the packed package on the server with `opsi-package-manager --install <package-file>`

10.2.6. Installing with opsi-package-manager

To install the packed product there is a command `opsi-package-manager`. To do this, go to the root directory of the product and execute the following command.

```
opsi-package-manager -i <myproductid_version.opsi>
```


10.2.7. Example of a 'control' file

```
[Package]
version: 1
depends:

[Product]
type: localboot
id: mytest
name: My Test
description: A test product
advice:
version: 3.14
priority: 10
licenseRequired: False
productClasses:
setupScript: setup.ins
uninstallScript:
updateScript:
alwaysScript:
onceScript:
customScript:
userLoginScript:

[ProductDependency]
action: setup
requiredProduct: javavm
requiredStatus: installed

[ProductProperty]
type: unicode
name: mytextprop
multivalue: False
editable: True
description: hint
values: ["off", "on"]
default: ["off"]

[ProductProperty]
type: bool
name: myboolprop
description: yes or no
default: False

[Changelog]
mytest (3.14-1) testing; urgency=low

* Initial package

-- jane doe <j.doe@opsi.org> Mi, 14 Jul 2010 12:47:53 +0000
```
10.2.8. Create opsi-package with CLI tool opsi-newprod

Do not use any country-specific symbols (umlaut), since the actual country code might vary for different code tables.

To start creating a new product, change directories to the product directory, and start the creation of the new product by entering the command `opsi-newprod`. The next question will ask you about the type of product you want to create. Choose the type `localboot` for products which should be installable by `opsi-client-agent/opsi-script`. The product type `netboot` is used for products which are activated as a bootimage (like OS installation).

![Figure 36. Choose the product type: localboot](image)

Confirm your choice with tab (or F12). Next, fill in the basic product parameters. At the top of the window there is an explanation for the current input field.

![Figure 37. Input of the product information](image)

**Product Id**

is a distinct short name for the product, independent from the product version (we recommend to use only plain ASCII letters and `-`, no white space, no special characters)

**Product name**

is the full name of the product

**Description**

is an additional description of the product.

**Advice**

is some additional information on how to handle the product (a note).

**Product version**

is the version of the packed software (max 32 chars).
Package Version

is the version of the package for the product version. For example, this helps to distinguish between packages with the same product version but with modified opsi-script scripts.

License required

is only relevant to netboot products.

Priority

controls the installation sequence. Possible Values are between 100 (at the very beginning) and -100 (at the end). Note: product dependencies also have influence on the installation sequence. See the opsi manual for more information.

After the product information is completed, fill in which action scripts should be provided:

![Figure 38. Input of the opsi-script script names for different actions](image)

After editing the product information you should mention the script you want to use for different activities.

Usually the Setup script is named setup.opsiscript

Usually the Uninstall script is named uninstall.opsiscript

An Update-Script will be used for minor changes on existing big installations. If this product is switched to the required action setup, then the update script will be automatically executed after the setup script.

An Always-Script will be executed at the beginning of every activity of opsi-client-agent (e.g. on every boot).

A Once-Script has the resulting state not_installed. It is a very special kind of script, and you should only use it if you really know what you are doing.

A Custom-Script doesn't change the resulting state. It is a very special kind of script, and you should only use it if you really know what you are doing.

A userLoginScript is used to modify the user's profile after the user logs into the system. It only works with the opsi extension User Profile Management, which is described at the User Profile Management chapter in the opsi-manual.

<table>
<thead>
<tr>
<th>Type</th>
<th>resulting state</th>
<th>resulting action</th>
</tr>
</thead>
<tbody>
<tr>
<td>setup</td>
<td>installed</td>
<td>none</td>
</tr>
<tr>
<td>uninstall</td>
<td>not_installed</td>
<td>none</td>
</tr>
<tr>
<td>update</td>
<td>installed</td>
<td>none</td>
</tr>
<tr>
<td>always</td>
<td>installed</td>
<td>always</td>
</tr>
<tr>
<td>once</td>
<td>not_installed</td>
<td>none</td>
</tr>
<tr>
<td>custom</td>
<td>unchanged</td>
<td>unchanged</td>
</tr>
<tr>
<td>User login</td>
<td>unchanged</td>
<td>unchanged</td>
</tr>
</tbody>
</table>
The next step is to define one or more product dependencies. If there are no product dependencies, select No.

![Create product dependency: No/Yes](image)

**Figure 39. Create product dependency: No/Yes**

To create a product dependency, enter the following data (help is available at the top of the window):

![Data needed to create a dependency](image)

**Figure 40. Data needed to create a dependency**

**Dependency for Action**
- Which product action shall the dependency create, or when should the dependency be checked (only setup).

**Required product id**
- Product id of the required product.

**Required action**
- Select the required action (setup) for the required product. If no required action is set, a required installation status must be set.

**Required installation status**
- Select the required status of the required product (installed). So the required product will be installed if it isn’t installed on the client yet. If no required installation status is set, a required action must be set.

**Requirement type**
- This is regarding the installation order. If the required product has to be installed before the installation of the actual product,
this is set to \textit{before}. If it has to be installed after the actual product, set requirement type to \textit{after}. Leave it blank if the installation order doesn't matter.

The possibility to define uninstall actions or dependencies is broken. After defining a product dependency, you will be asked if you want to create another product dependency. If you choose Yes, then the procedure for defining a product dependency is repeated. If you choose No, then you will be asked to define some product properties, which means defining additional switches for product customization.

The installation sequence results from a combination of product dependencies and product priorities. For details on how this is done, and what you can configure, see the opsi-manual.

\textbf{Figure 41. (another) product property to create?}

If you answer Yes, you will have to describe the product properties.

The product properties are client specific, and have names (keys) which can hold different values. These values can be evaluated by the \textit{opsi-script} script, and result in installing different options at installation time.

First we have to decide if our property is a text value (\textit{unicode}) or a logical value e.g. true/false (\textit{boolean}). If you are not sure choose \textit{unicode}.

\textbf{Figure 42. Choose the data type of the property}

Next, a description for the switch needs to be specified. This description will be shown in the opsi-configed as a help text. Next, you can define the set of values for the switch (separated by comma). If this is left blank, then any value is allowed for the switch.

If a values contains a backslash \textbackslash it has to be doubled.

An example showing how a path would be defined: C:\\temp
Figure 43. Description of the product properties

Next, you can decide if the product property has a default value (switch).

Figure 44. Default value of the product property

If you choose boolean as the data type, then the description will contain only the Property name and Property description.

Figure 45. Description of a boolean property

After defining a product property, you will be asked if you want to create another product property. If you choose Yes, then the procedure of defining a property will be repeated. If you choose No, then you will be asked for name and email of the product maintainer. This data will be written on the changelog.

Figure 46. Input of the maintainer data

Finally, the basic definitions for the new product are done.

Using the list command (ls), you can see the directory structure as described above. Change to the OPSI folder and list the content. The control file now contains the data you just defined, and you can load the file into an editor to view or change the entries.

10.3. Suggestions on How to Solve Problems with opsi-script Scripts

10.3.1. Installation When the User is Logged on

Before we begin, we assume that you have tried an unattended installation using an opsi-script script, and the installation worked OK when the user had administrative privileges. However with some software products, you will see that the installation fails when started from within the opsi deployment software (opsi-client-agent). A possible reason for that difference might be that the installation process requires knowledge about the user environment or profile.
In the case of a MSI package, the option 'ALLUSERS=1' might help. Example:

```plaintext
[Actions]
DefVar $MsiLogFile$
Set $MsiLogFile$ = %opsiLogDir% + "\myproduct.log"
winbatch_install_myproduct

[winbatch_install_myproduct]
msiexec /qb-! /l*$MsiLogFile$ /i "%ScriptPath%/files\myproduct.msi" ALLUSERS=1
```

Another possibility is that the installation starts a second process and stops before the second process is finished. So from the point of view of the opsi-script script, the task is finished while in fact the second process is still working (installing / uninstalling).

In this case, you may use the modifier /WaitSeconds <seconds>, or /WaitForProcessEnding "program.exe" /TimeOutSeconds "<seconds>", in the WinBatch section so that the script waits for the end of the second process.

Another more complex way to solve the problem is to create a temporary administrative user account and use this account for the program installation. For a detailed description on how to do this, please refer to the opsi-script manual chapter 8.3 'Script for installation in the context of a local administrator':


### 10.3.2. Customization after a silent/unattended Installation

After a successful silent installation, some customizing might be useful. The opsi-script is a powerful tool to do that job. First, find out what patches have to be applied. For example, that could mean analyzing which registry settings are affected by the GUI customizing tools.

You can use the tools shown in [opsi-getting-started-softwintegration-tutorial-analyse-and-repackage]. Some other tools can be found here:

Some other often used tools are:

- sysinternals
- regshot

### 10.3.3. Integration with Automated Answers for the setup Program

Another fast way of integration is to provide an automated answer file for the setup process. The answer file contains pre-defined answers. To be more precise, the answer file is used by a control tool, which waits for the setup to come up with the interactive windows. The control tool then passes input to these windows as defined in the answer file. As a control tool we recommend 'AutoIt'. The AutoIt program and the documentation can be found at: [http://www.hiddensoft.com/autoit3](http://www.hiddensoft.com/autoit3).

AutoIt provides a lot of commands to control the setup process. Also, several error states can be handled (if known in advance) with the '[ADLIB]' section in the script.

There is, however, a fundamental challenge in using AutoIt:

The AutoIt script must provide input for every window that might pop up during installation. So if any unexpected window pops up, which isn't handled in the [ADLIB] section, AutoIt provides no input for this window and the installation stops at that point while waiting for input. This input could be done interactively by a user, and then the script can take over again and handle the next windows.

Another situation that may cause failure of an AutoIt installation:

The user can interfere with the installation if the mouse and keyboard are not disabled. Therefore we regard 'unattended' or 'silent' setup as a more stable solution.

A combination of both might do a good job:

The 'silent'-setup does the main installation and the AutoIt script handles special conditions that might occur.
If you use the opsi option of running the installation on another desktop than the current desktop, or if the current desktop is locked, then you will find that some autoit functions do not work properly under these conditions.

Therefore you should avoid using the following autoit commands in ‘opsi-script’ scripts:

- winwait()
- winactivate()
- Send()

These are the most widely used commands.

We recommend to use the opsi-autoit-lib.au3 library, that provides replacements for this commands. You will find this file at C:\Program Files (x86)\opsi.org\opsi-client-agent\opsi-script\lib\opsi-autoit-lib.au3.

This library provides substitutes:

**winwait()**

should be replaced by the function

opsiwinwait($title, $text, $maxseconds, $logname)

**Send()**

should be replaced by the function

opsiControlClick($title, $text, $id, $maxseconds, $logname)

respectively by

opsiControlSetText($title, $text, $id,$sendtext, $maxseconds, $logname)

It is always a good idea for the identification of controls to use the program Au3info.exe to get the 'ControlId' needed by these commands. Please use the numerical 'ControlId', because the other variants do not seem to work properly.

Example you will find in: C:\Program Files (x86)\opsi.org\opsi-client-agent\opsi-script\lib\ in den Dateien autoit_example_1.txt and autoit_example_1.txt.

see also:

http://www.autoitscript.com/wiki/FAQ#Why_doesn.27t_my_script_work_on_a_locked_workstation.

http://www.autoitscript.com/autoit3/docs/

http://www.autoitscript.com/autoit3/docs/intro/controls.htm

http://www.autoitscript.com/autoit3/docs/functions.htm

**10.3.4. Analyze and Repackage**

When a software developer builds a setup for deployment, the developer usually knows about the required components of the software that have to be installed. But if somebody has a black box as a setup, then they need first to analyze what the setup does. This can be done by monitoring the setup activities with the appropriate tools (e.g. monitoring files and registry access) or by comparing the system states before and after installation.

To analyze the before or after states, there are several tools. For Example:

- regshot
- appdeploy-repackager

**10.3.5. How to uninstall Products**

To uninstall a software product from a computer, you need an 'uninstall' script to perform the deletion. The fundamental difficulty in software deletion is deciding what exactly has to be removed. Not all of the files that came with a software package can be deleted afterwards. Sometimes a package comes with standard modules that are also referred to by other programs. Often only the software manufacturer himself knows what parts have to be removed. The manufacturer's setup might offer an unattended uninstall option which can be embedded in the opsi uninstall script. Otherwise opsi-script provides several commands for software deletion:
Using an uninstall routine

If the product manufacturer provides an option for software deletion, you must check whether or not it can be run unattended (or in silent mode). If it requires some user interaction, an AutoIt script combined with the uninstall routine might do the job. The uninstall statement can be embedded in a [WinBatch] section of the opsi-script script:

```plaintext
[WinBatch_start_ThunderbirdUninstall]
"%SystemRoot%\UninstallThunderbird.exe" /ma
```

When using an uninstall program, always run a test to see if all of the files have been deleted and the computer is still in a stable state.

Products that are installed by MSI normally come with an uninstall option, which is usually the program `msiexec.exe` combined with the parameter `/x`. The parameter `/qb-!` is for the unattended mode (or without user interaction). So here is an example of an unattended uninstall command:

```plaintext
msiexec.exe /x some.msi /qb-! REBOOT=ReallySuppress
```

Instead of the package name, you could also use the GUID (Global Unique ID) with `msiexec.exe`. This GUID identifies the product in the system, which can be found in the registry directory ‘HKLM\Software\Microsoft\Windows\CurrentVersion\Uninstall’.

A request using the GUID looks like this:

```plaintext
msiexec.exe /x {003C5074-EB37-4A75-AC4B-F5394E08B4DD} /qb-!
```

If none of these methods are available or sufficient, the uninstall can be done using a opsi-script script as described below:

**Useful opsi-script commands for uninstall**

If a product has been installed by opsi-script functions, or if there is no uninstall routine for the product, the complete uninstall has to be done by a opsi-script script. opsi-script comes with some powerful uninstall functions. This chapter provides a brief overview of the uninstall functions, and more detailed information can be found in the opsi-script handbook.

Basic uninstall means deleting one or more files from the file system. This command can be executed from a opsi-script files section:

```plaintext
delete -f <file name>
```

or to delete a directory including sub directories:

```plaintext
delete -sf <dir name>\
```

The parameter ‘f’ means ‘force’ or to delete the files even if they are marked as ‘read only’ and the parameter ‘s’ means including the ‘subdirectories’. A file or directory can be deleted from all user profiles using the option ‘/AllNTUserProfiles’ (see opsi-script manual for details).

Directories containing files with the attribute ‘hidden’ or ‘system’ can be deleted by using a ‘DosInAnIcon’-section:

```plaintext
[DosInAnIcon_deleteDir]
rdir /S /Q "<List>"
```
To stop a running process before deletion use the \textit{killtask} command with the process' name (look at the task manager for process name):

\begin{verbatim}
KillTask "thunderbird.exe"
\end{verbatim}

If the product or part of it, runs as a service, you will have to stop the service before deleting the files. One way to do so, is to set the service state to inactive in the registry and restart the computer. Or to stop the service by using the command 'net stop', which doesn't need a reboot:

\begin{verbatim}
net stop <servicename>
\end{verbatim}

Deleting DLL files also requires special attention, since DLLs could also be used by other products. There is no general way of handling this.

To delete registry entries with opsi-script you can use the command \texttt{DeleteVar}. This command deletes entries from the currently open key:

\begin{verbatim}
DeleteVar <VarName>
\end{verbatim}

To delete a registry key with all sub keys and registry variables, you can use the opsi-script command \texttt{DeleteKey}:

\begin{verbatim}
DeleteKey [HKLM\Software\Macromedia]
\end{verbatim}

\section*{10.3.6. Known Issues with the 64 Bit Support}

The opsi installer opsi-script is a 32 bit program. There are no known problems when installing 32 bit software on a 64 bit system using opsi-script. For the installation of 64 bit software, some constants (like \texttt{%ProgramFilesDir%}) give wrong values.

opsi-script have special commands to handle these problems. So read the \texttt{opsi-script manual} for these issues.
11. General information about Windows

11.1. opsi directories on Windows

Important opsi directories and files on the Windows client:

- `c:\program files (x86)\opsi.org\opsi-client-agent`
- `c:\opsi.org (other opsi logfiles and variable files)`
12. Further information

The opsi manual contains a wealth of further information that is important for usage in production. If you use opsi in production, we particularly recommend you to familiarize yourself with the 'opsi-backup' tool to create a backup of your data.

If you do not find it there or need help, contact the opsi community.

For installations in production we recommend professional support by uib with a maintenance and support contract.